Burning Issues with *Prometheus* - the Canadian Wildland Fire Growth Simulation Model

J. Barber* C. Bose† A. Bourlioux‡ J. Braun§ E. Brunelle‡
T. Garcia¶ T. Hillen‖ B. Ong**

September 11, 2009

Abstract: The software package *Prometheus* is Canada's only operational wildfire growth modelling simulator. The program calculates accurate, fast, multi-day forecasts of moving fire fronts and is currently used in fire fighting situations, in fire risk analysis and in the design of "fire-safe" communities and forests. At the core of *Prometheus* is an algorithm that calculates the evolution of the fire front in an empirical manner, assuming locally elliptic fire spread and Huygens' principle in a Lagrangian description of the evolution.

This report describes progress made on a set of problems that *Prometheus* developers brought to the 10th Pacific Institute for the Mathematical Sciences (PIMS) Industrial Problem Solving Workshop held in June 2006 at Simon Fraser University.

In particular, we investigate the mathematical background behind the *Prometheus* code and suggest a new method (the "outer hull" approach) to remove tangles from the evolving front. Other methods that can reduce tangles and crossings are "smoothing" of data and parameters, and redistribution of vertices on the evolving front. We investigate the use of de Boor's Algorithm as an automated procedure for redistributing vertices along the front. Finally, we investigate the level set method as a new tool for forest fire spread simulations and show in some simple test cases the excellent potential of this method.

1 Introduction

Prometheus was conceived in 1999 and developed over intervening years into a fully operational,

*Mathematics, The University of Arizona
†Mathematics and Statistics, University of Victoria
‡Mathématiques et Statistique, Université de Montréal
§Statistical and Actuarial Science, University of Western Ontario
¶Dept. of Statistics, Texas A & M
‖Mathematical and Statistical Sciences, University of Alberta
**Mathematics, Simon Fraser University

essentially real-time[1] fire spread simulator. The computational rules for moving the active fire front are based on theoretical work of Richards [3], and Richards, Bryce [8]. This theory assumes elliptical growth of a fire front in homogeneous conditions (eccentricity and orientation of axes based on wind and slope conditions), as described in the Canadian Forest Service Fire Behavior Prediction (CFSFBP) database. Under varying spatial conditions, the above assumption is combined with Hugyens' principle, treating the fire front as an infinite collection of microscopic, independent elliptical fires (see Section 1.2 for details). This approach is very natural, and the implementation in *Prometheus* has earned a reputation for giving good results in the field.
The industrial scientists who brought *Prometheus* to the PIMS Industrial Problem Solving Workshop identified several interesting problems that they would like solved and incorporated into future software versions:

- The fire front can develop complicated knots and crossovers which adversely affect secondary calculations in the program and lead to unrealistic fire fronts. *Prometheus* developers would like an automated 'untangler' routine to remove these computational artifacts at each time-step in the calculation. We propose various untangling strategies including an efficient "outer-hull" approach in Section 2, smoothing in Section 3, and redistribution of vertices in Section 4.

- *Prometheus* contains a number of threshold parameters used to stabilize the computation and provide for varying degrees of regularization of the evolving front. Currently these parameters are user-tuned which can naturally lead to operator error. The developers would like algorithms that automatically adjust these parameters as the computation proceeds. We present some ideas in Section 3.

- While current verstions of *Prometheus* are stable, well-respected and widely adopted fire simulation tools, the developers are interested in other computational approaches that might equal, or possibly exceed the performance of the marker method approach. We introduce the level set method for forest fire spread in Section 5. The level set method allows for a simple and efficient inclusion of obstacles, fire barriers and topological changes. The problem of tangles and crossings does not arise within the level set formulation. Initial computations show that the level set method is computationally efficient and accurate.

1.1 Acknowledgements

While the work presented in this paper was initiated during the Tenth PIMS Industrial Problem Solving Workshop, held at Simon Fraser University during the week of June 26, 2006, ongoing interest in the problem by a number of workshop participants after the formal workshop led to further progress which has been incorporated into this extended report.
The industrial problem entitled *Prometheus – the Canadian Wildland Fire Growth Model* was submitted, presented and represented during the workshop by *Prometheus* scientists, Cordy Tymstra and Robert Bryce who patiently answered our many questions about the software and the fire propagation problem. The authors are pleased to acknowledge the Pacific Institute for

[1]Run times for *Prometheus* simulations are fast enough to provide multiple fire scenarios, based on current and forecast conditions to forest fire managers and fire boss' during fire fighting events.

the Mathematical Sciences (PIMS) and the Alberta Sustainable Resource Development Branch (under which the *Prometheus* project has been hosted since inception) for support on this project, and SFU for hosting the workshop.
It would be difficult to make a complete list of all the mathematical scientists who may have made contributions on the *Prometheus* problem during the workshop; it is a unique feature of these events that individuals circulate amongst various workshop teams, listening, asking questions and making contributions as they feel appropriate. However, we are happy to acknowledge particularly two individuals, Clarence Lam (Concordia) and C. Pöschl, (Innsbruck) who were steady members of our *Prometheus* team during the week.
This IPSW project developed into a full MITACS Project on "Forest Fires and Spread in Heterogeneous Landscapes", where many of the original ideas are being further developed. The work of CB, AB, JB and TH is supported through the MITACS project and through individual NSERC grants.

1.2 Huygens' Principle and the Model Equations

Richards [8], [2], [3] derived a differential equations model for fire propagation. The model is *emperical* in the sense that it takes the experimentally derived CFSFBP spread rates as the main inputs, rather than attempting to model the detailed physical processes involved in combustion, thermal aspects near the fire front and the complex coupling of these processes with the atmosphere. Models of the latter type are termed *physical models* and while they are interesting and challenging from a mathematical point of view, they are extremely difficult to set up and require extreme computational resources; no physical models have been developed into operational models at this time. Therefore, continued work on empirical models can be justified on both a mathematical and a practical basis.
It is important to note that Richards' model is inherently set in a two-dimensional spatial domain and takes into account fuel, weather and topographical conditions. Three dimensional effects (slope and topography) are incorporated into the two-dimensional model using a projection method. This way, one views the spread of the fire as if it were evolving on a topographical map of the terrain, with the slope accounted for by the non-homogeneity in the model parameters with respect to the spatial domain.
Assuming a locally smooth fire front and using Huygens' principle, the model states that the position of the next fire front is the envelope of small ellipses generated in a finite time step starting at each point of the current front; this evolution is defined by Richards' set of differential equations.
Richards' derivation of these equations begins by assuming that under constant conditions for homogeneous fuels, slope and wind, a fire ignited at a point will expand at a constant rate as an ellipse of the form:

$$\begin{aligned} x(s,t) &= bt\cos s \\ y(s,t) &= at\sin s + ct \end{aligned} \tag{1}$$

where t is time, s is a counter clockwise angle with respect to the horizontal line at $y = ct$ as in Figure 1 the point of ignition is assumed at the origin, and the wind direction is considered to be

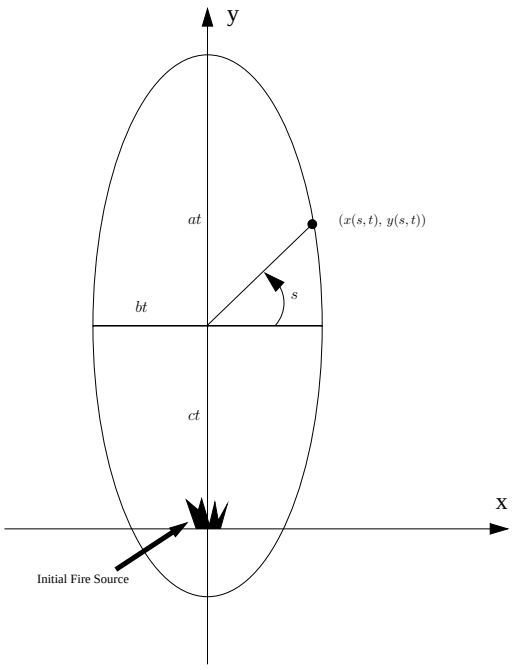


Figure 1: The elliptical shape of fire growth from a point at time t.

the positive y-axis. Define the Rate of Spread ROS $= a+c$, the Flank Rate of Spread FROS $= b$ and the Back Rate of Spread BROS $= a-c$. The quantities ROS, FROS, and BROS are exactly the parameters available from the CFSFBP database. They are dependent on local conditions such as fuel type, temperature and moisture content; they are the fundamental inputs for the empirical fire front spread. Hence we should write parameters a, b, c as: $a(x,y), b(x,y), c(x,y)$. In fact some long range models should allow the parameters to depend on time as well, but for simplicity, in this report we will be content with just spatial dependence. One other input parameter RAZ $= \theta$ is the effective wind direction, measured with respect to the positve $y-$axis. RAZ is a combination of ambient wind and the effect of slope (there is an observed tendency of fire to spread faster uphill, all other things being equal). In our simplified elliptic model above, for example, we have taken, RAZ $= 0$.

In order to take into account the spatial variation in these input parameters, Richards invokes Hugyens' principle by viewing the active fire front at time t as an infinite collection of ignition points, each of which evolves independently according to Equation 1 with local parameter values, and over a small time interval dt. The new fire front at time $t + dt$ is declared to be the 'outer hull' of these small elliptical fires at time dt. We now briefly outline the details of this approach.

Assume we can represent the coordinates of the fire front at time t by $(x(s,t), y(s,t))$, where $0 \leq s \leq S$ is a parameter along the curve, for example the arc length. Then, the small elliptical fire generated by the point $(x(s,t), y(s,t))$ has the angle $RAZ = \theta$ and is defined by equation (1) where $t = dt$ and a, b, and c are defined by the fuel, wind and topographical conditions at that point. Take dt small enough so that a, b, and c remain constant for the time period. Thus, the new fire front at time $t+dt$ will be the outer envelope of the ellipses generated at each point on the curve at time t. So, given $(x(s,t), y(s,t))$, $0 \leq s \leq S$, the curve $(x(s,t+dt), y(s,t+dt))$,

$0 \leq s \leq S$, is defined for positive dt.
A key idea, introduced by Richards in [3] is to calculate calculate $(x(s,t+dt), y(s,t+dt))$ by first transforming the ellipses into circles via a linear transformation T. This will allow the computation of $(x(s,t+dt), y(s,t+dt))$ by a simple geometric construction which we detail below.
The linear transformation T will first rotate the axes so that the positive $y-$ direction is that of the wind, and then scale the x coordinate by $a(s,t)/b(s,t)$ to eliminate the eccentricity of the elliptical shape; the circles now have radius $dt \cdot a(s,t)$ and centers at a distance $dt \cdot c(s,t)$ above the point generating them. Defining

$$\Gamma = \begin{pmatrix} a/b & 0 \\ 0 & 1 \end{pmatrix} \tag{2}$$
$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

we have

$$T = \Gamma \cdot R_\theta. \tag{3}$$

We remind the reader that all of the above parameters defining the transformation T are spatially dependent, in particular, they can be expressed as functions of s and t.
Applying T to the elliptical coordinates $(x(s,t+dt), y(s,t+dt))$, we obtain the circular coordinates X, Y:

$$X = [a(s,t)/b(s,t)](x\cos\theta - y\sin\theta) \tag{4}$$
$$Y = x\sin\theta + y\cos\theta.$$

To find the coordinates $(X(s,t+dt), Y(s,t+dt))$ we first draw two circles at time t at points s and $s+ds$, where the circle at $s+ds$ has radius $dt \cdot a(s+ds,t)$ and its center a distance $dt \cdot c(s+ds,t)$ above the generating point. The coordinates of the circle at $s+ds$ can be found using simple geometry and truncated Taylor series (since we will be taking the limit in ds the higher terms will tend to zero). Once these coordinates are obtained, let ds tend to zero to obtain the coordinates $(X(s,t+dt), Y(s,t+dt))$ as:

$$\begin{aligned} X(s,t+dt) &= X(s,t) + \frac{dta(-X_s dt\cdot a_s + (dt\cdot c_s + Y_s)((dt\cdot c_s+Y_s)^2 + X_s^2 - dt^2 a_s^2)^{1/2})}{((dt\cdot c_s+Y_s)^2+X_s^2)} \\ Y(s,t+dt) &= Y(s,t) + \frac{dt\cdot a(-((dt\cdot c_s+Y_s)^2+X_s^2-dt^2a_s^2)^{1/2}X_s - (dt\cdot c_s+Y_s)^2 dt\cdot a_s)}{((dt\cdot c_s+Y_s)^2+X_s^2)} + cdt \end{aligned}$$

These equations define the envelope of the circles formed in finite time dt (see [3]).
Taking the inverse transformation, T^{-1}, dividing by dt and letting dt tend to zero, we obtain Richards' differential equations for spread of the fire front:

$$\begin{aligned} x_t(s,t) &= \frac{b^2\cos\theta(x_s\sin\theta + y_s\cos\theta) - a^2\sin\theta(x_s\cos\theta - y_s\sin\theta)}{\sqrt{a^2(x_s\cos\theta - y_s\sin\theta)^2 + b^2(x_s\sin\theta + y_s\cos\theta)^2}} + c\sin\theta \\ y_t(s,t) &= \frac{-b^2\sin\theta(x_s\sin\theta + y_s\cos\theta) - a^2\cos\theta(x_s\cos\theta - y_s\sin\theta)}{\sqrt{a^2(x_s\cos\theta - y_s\sin\theta)^2 + b^2(x_s\sin\theta + y_s\cos\theta)^2}} + c\cos\theta \end{aligned} \tag{5}$$

subject to the initial conditions

$$\begin{aligned} x(s,0) &= x_0(s) \\ y(s,0) &= y_0(s). \end{aligned} \tag{6}$$

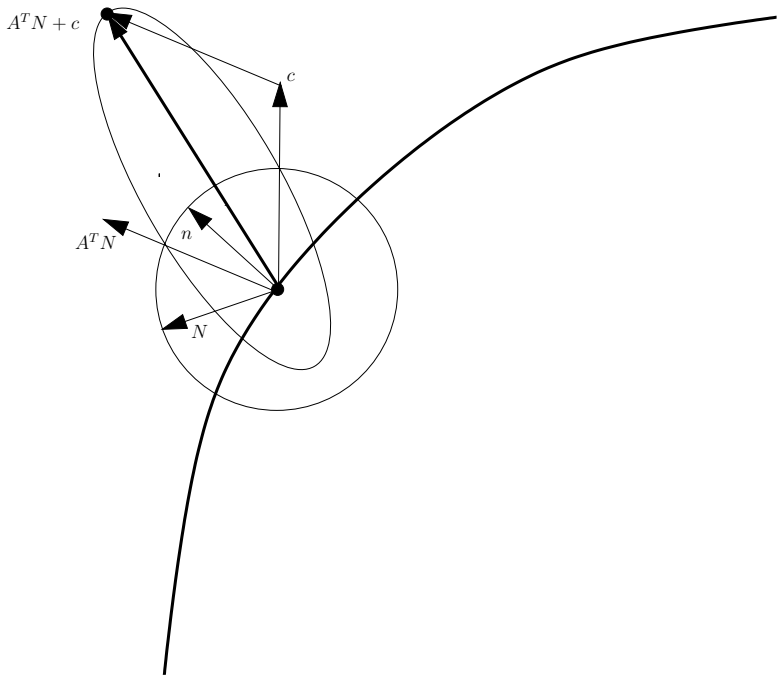


Figure 2: The graphical view of the reduced mathematical formulation for the differential spread equations.

Note that while these equations are consistent with the presentation given in [10], there is an exchange of the parameters a and b when compared to the original paper of Richards [3].
Since it is difficult to see the simple geometric derivation underlaying the spread equations directly from the form (5) so we end this section by deriving a simplified form.
Let $A = \begin{pmatrix} b & 0 \\ 0 & a \end{pmatrix} \cdot R_\theta$, $\overrightarrow{n} = (y_s, -x_s)$, $\overrightarrow{c} = R_\theta^T(0, c)^T$.
Then the equations in (5) can be rewritten as:

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \frac{A^T A \overrightarrow{n}}{\|A\overrightarrow{n}\|} + \overrightarrow{c} \tag{7}$$

Notice R_θ is an orthogonal matrix, so $R_\theta^{-1} = R_\theta^T$.
We can see this visually in Figure 2. In this figure, N denotes $\frac{A\overrightarrow{n}}{\|A\overrightarrow{n}\|}$.

1.3 The Marker Method for Front Propagation

The equations in (5) and (7) are solved numerically in the *Prometheus* model using the so-called *marker method* or *Lagrangian formulation* which we now describe.
We discretize the curve $(x(s,t), y(s,t))$ into n vertices and connecting line segments between vertices, where

$$P_i^j = (x_{i,j}, y_{i,j}) = (x(ids, jdt), y(ids, jdt)) \tag{8}$$

are the vertices (j representing the current time step and i indexes the vertex). For n vertices we have $ds = S/n$, and dt is the time step size. Fire fronts in *Prometheus* are always assumed

to be closed curves, so $P_0^j = (x_{0,j}, y_{0,j}) = (x_{n,j}, y_{n,j}) = P_n^j$ closing the polygonal approximation to the curve.
We link this discrtization to the the spread equations in (5) by using a centered-difference in s and forward difference in time in order to solve for $(x_{i,j+1}, y_{i,j+1})$. If we let F and G denote the right sides of the two equations in x and y respectively in (5) we obtain

$$\begin{aligned} dx_{i,j} &= dtF(ids, jdt, (x_{i+1,j} - x_{i-1,j})/2ds, (y_{i+1,j} - y_{i-1,j})/2ds) \\ dy_{i,j} &= dtG(ids, jdt, (x_{i+1,j} - x_{i-1,j})/2ds, (y_{i+1,j} - y_{i-1,j})/2ds) \end{aligned} \tag{9}$$

and hence:

$$\begin{aligned} x_{i,j+1} &= x_{i,j} + dx_{i,j} \\ y_{i,j+1} &= y_{i,j} + dy_{i,j} \end{aligned} \tag{10}$$

In the numerical simulation, parameters a, b, c and θ contained in the functions F and G are stored as gridded data (exactly as available from the CFSFBP system and derived from known topographical and weather forcast data) and accessed for Equation 9 through a lookup table.
Translating these into the notation of vertices $P_i^j = ((P_i^j)_x, (P_i^j)_y)$ we obtain the following difference equations, as implemented in *Prometheus*.

$$\begin{aligned} (\Delta P_i^j)_x &= \Delta t \frac{b^2 \cos\theta(x_s \sin\theta + y_s \cos\theta) - a^2 \sin\theta(x_s \cos\theta - y_s \sin\theta)}{\sqrt{a^2(x_s \cos\theta - y_s \sin\theta)^2 + b^2(x_s \sin\theta + y_s \cos\theta)^2}} + c\Delta t \sin\theta \\ (\Delta P_i^j)_y &= \Delta t \frac{-b^2 \sin\theta(x_s \sin\theta + y_s \cos\theta) - a^2 \cos\theta(x_s \cos\theta - y_s \sin\theta)}{\sqrt{a^2(x_s \cos\theta - y_s \sin\theta)^2 + b^2(x_s \sin\theta + y_s \cos\theta)^2}} + c\Delta t \cos\theta \end{aligned} \tag{11}$$

$$\begin{aligned} (P_i^{j+1})_x &= (P_i^j)_x + (\Delta P_i^j)_x \\ (P_i^{j+1})_y &= (P_i^j)_y + (\Delta P_i^j)_y \end{aligned}$$

where

$$\begin{aligned} a = \frac{ROS_j^i + BROS_j^i}{2}, \quad b = FROS_j^i, \quad c = \frac{ROS_j^i - BROS_j^i}{2} \\ \theta = RAZ_j^i, \quad 2x_s ds = P_{x_j}^{i+1} - P_{x_j}^{i-1}, \quad 2y_s ds = P_{y_j}^{i+1} - P_{y_j}^{i-1} \end{aligned} \tag{12}$$

These are the discrete versions of the spread equation (5) as computed in the *Prometheus* code. As a quick check for consistency of these equations with the continuous version in (5), observe that solving for ROS in (12) above, for fixed i and j, yields forward rate ROS_j^i as $a + c$, and flank rate FROS_j^i as b.
Initial conditions (the front position and shape at time $t = 0$ corresponding to $j = 0$) are user inputs; A typical choice that is a default configuration in *Prometheus* is a circle of 32 vertices ordered counter clockwise, with its radius 1/5th the data grid-cell size.

2 Untangling and Other Topological Considerations

In this section we consider the problem of knots, crossovers and loops which develop as the *Prometheus* engine propagates a fire front using the discrete marker method and localized propagation data. This is a well-known problem with the marker approach and the Lagrangian

formulation – various topological 'fixes' have been discussed in the literature under terms like *delooping* or *clipping*; Chapter 4 of the book [6] gives a good overview of the issues involved. See also [5] where delooping methods for a 2-D photo etching process is developed. The accumulated experience as reported in this literature is that 2-dimensional untangling is complicated, but feasible for specific models, however for 3 or more space dimensions the topological problems become unmanageable. Keeping in mind that the *Prometheus* model is inherently 2-D, we consider a number of approaches that could improve the stability and, we hope, lead to a more automated untangling routine for the specific problems encountered by *Prometheus*.

In this section we first describe the current approach used by *Prometheus*: a winding number calculation followed by heuristics for untangling. We found that the two numerical methods (*scan line* vs. the *Bryce-Richards algorithm*), previously thought to be equivalent, can produce different results even for fairly simple fronts. Worse, both methods can fail to correctly determine the state of a vertex on a knotted fire front.

Next we describe Parameter smoothing (detailed in Section 3) as a way to avoid crosssovers which arise from discontinuities in the discretized spatial data. Where such crossovers cannot be avoided, we propose a completely new method for identifying active segments on the fire front, and for removing inactive ones based on sequential resolution of crossings and knowledge and identification of the outer hull. Finally, the tendency of evolving vertices to cluster, or to become sparse on the fire front can increase the frequency of crossings. Some sort of (*regridding* or *redistibution*) of the vertex set seems to be necessary[2] In Section 4 we discuss a powerful approach for vertex redistribution via *monitor functions* and *de Boor's Algorithm*. 4.

2.1 The Existing Approach to Knots and Crossovers

As discussed in section 1.3, the *Prometheus* model uses the Marker method to depict fire spread. At each time-step, the fire front is depicted by a set of vertices connected by vectors in counter clockwise order. Each vertex is evaluated for being either *active* or *inactive*, and only *active* vertices should be allowed to propagate the subsequent fire front using equations (11) and (12). Ideally, active vertices are ones shaping the fire front and burning into unburned, fuel rich areas, and inactive ones are in already burned areas. However, because of the calculation methods for the fire front, crossover of vertices or edges occur, leading to active vertices propagating into already burning regions or inactive vertices in fueled regions where they should be burning. To improve the *Prometheus* model, we would like to remove these instances. Thus, an accurate method for determining the active/inactive state of each vertex is necessary.

In *Prometheus*, the turning number of each vertex determines the vertex's state of activity. By definition, "the turning number of a vertex is the number of times a particle traversing a directed path around the curve will rotate counter clockwise around the vertex, (with clockwise rotations cancelling counter clockwise rotations), and will be zero if and only if the vertex is external to the curve" [8]. The curve divides the plane into regions of equal turning number, where regions of zero turning number are external to the curve.

Two methods have been presented to evaluate the turning number of each vertex:

[2] In fact, the present *Prometheus* has a number of heuristic rules for adding new vertices to the evolving front and for freezing evolution of other vertices.

1. Bryce-Richards algorithm [8].
2. Scan Line [13].

Although both algorithms agree with each other most of the time, there are some instances where discrepancies occur. These are now discussed in further detail.
Both methods assume each vector in the polygonal path to have an upward or downward orientation, that is, strictly positive (upward) or strictly negative (downward) second component. If this is not the case a small perturbation in adjacent vertices is made to eliminate the horizontal vector. In fact, the current algorithm adjusts all horizontal vectors to give them an upward orientation.

2.1.1 Bryce-Richards Algorithm

In the Bryce-Richards algorithm, we say a vertex is *active* if it is adjacent to a region of zero turning number. Assuming that each vertex has only one incoming vector and one outgoing vector, each vertex is adjacent to two regions of different turning number; one of these regions is considered to be left of the vertex, and the second is either to the right, above, or below. We determine the turning number of each region as follows.
Call the vertex to be evaluated vertex A. First, a horizontal line segment L is drawn from vertex A to a position left of the entire curve. The turning number of the left hand region adjacent to A is defined to be the number of downward crossings of L by the curve, minus the number of upward crossings of L by the curve. The intersection of L with A is not included.
If L passes through another vertex, call it B, we make the following calculations:

a. If both vectors adjacent to B are of the same orientation, then a single intersection of this orientation is said to have occurred.

b. If both vectors adjacent to B are of different orientation, then no intersection is considered to have occurred.

If the turning number of the left hand region is zero, then vertex A is said to be *active* and the process is complete. Otherwise, we must evaluate the turning number of the second region.
The turning number of the second region is determined by including the intersection of the horizontal line L at vertex A in the difference between the number of downward and upward intersections. We proceed as follows:

a. If both vectors adjacent to A are of the same orientation, then a single intersection of this orientation is said to have occurred. Also, the second region is labeled as being right of the first region.

b. If both vectors adjacent to A are of different orientation, then an intersection with an orientation equal to that of the left hand vector is said to have occurred. If both adjacent vectors to A are above it, then the left hand vector is the one that induces the largest counter clockwise angle to the x-axis. Also, the second region is deemed to be above or below A.

If the turning number of the second region is zero, vertex A is *active*; otherwise, it is *inactive*.

2.1.2 Scan Line Approach

Another method for determining the turning number of a vertex is via a *Scan Line* described by Robert Bryce during the workshop. In this approach, one first perturbs the location of each vertex: following the curve in a counterclockwise manner, move each vertex an $\epsilon > 0$ distance to the right relative to the direction of the curve one is traversing. For each perturbed vertex, draw a horizontal line through that vertex that extends to the outside edges of the entire curve. Starting from either the left or right end of the horizontal line, the turning number of each region of the curve is determined as follows:

a. Regions outside the entire curve have turning number zero.

b. When the horizontal line crosses through a downward arrow add one.

c. When the horizontal line crosses through an upward arrow subtract one.

The turning number of the vertex being evaluated is the turning number of the region where its perturbed vertex lies.

2.1.3 Discrepancy between the Two Methods

It was initially believed that both methods described above would produce the same results. However, after a closer examination, we see this cannot be true in all situations because in the Bryce-Richards algorithm, the turning number is determined by two regions (and only one has to have zero turning number to mark the vertex *active*), whereas the *Scan Line* only considers the one region where the vertex lies.
The following figures illustrate the discrepancies between the two methods. Among the figures, we have included an example previously constructed by Richards and Bryce in [8]; this example (see Figure 4) instigated the discussion of the discrepancies observed. All the figures drawn here are the result of an R program [14] simulation of both algorithms on the examples.
In all figures, we denote a red dot as an *active* vertex, a black dot as an *inactive* vertex, a purple dot as the perturbed vertex, red arrows are vectors with an upward orientation, blue arrows are vectors with a downward orientation, and the green horizontal line represents a scan line. The black numbers in each figure represent the turning number of each region.

Example 1: Figure 3 was designed as a way of understanding the *Scan Line* approach. In reality, it can be generated when a fire crosses over itself at the tip. In this figure, we see that after vertex A is perturbed to vertex A^*, it is in a region with turning number 1 and hence, by the *Scan Line* the turning number of vertex A is 1 and thus, is *inactive*.
However, our results differ when we implement the Bryce-Richards algorithm. Following the algorithm, we first determine the turning number of the left hand region of A. When doing so, we find that the left hand region has turning number 1. Because this is non-zero, we must find the turning number of the second region. To do so, we include the direction of the left hand vector adjacent to vertex A which is, in this case, a downward direction. Hence, the Bryce-Richards algorithm finds the turning number for vertex A to be 2 and thus, *inactive*.

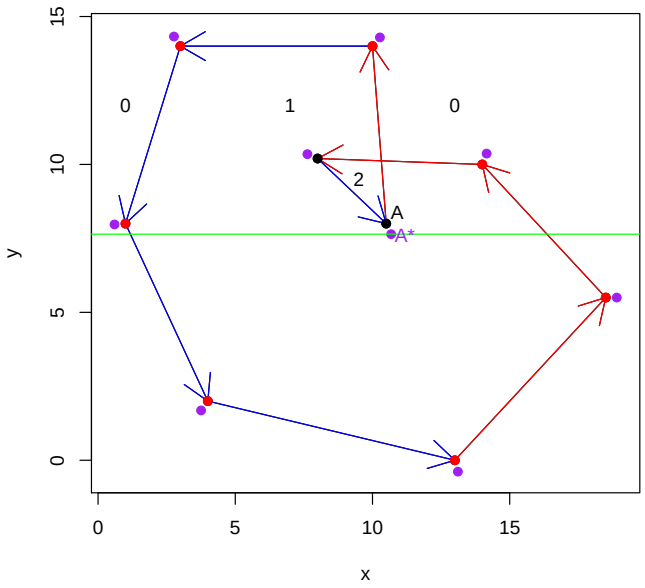
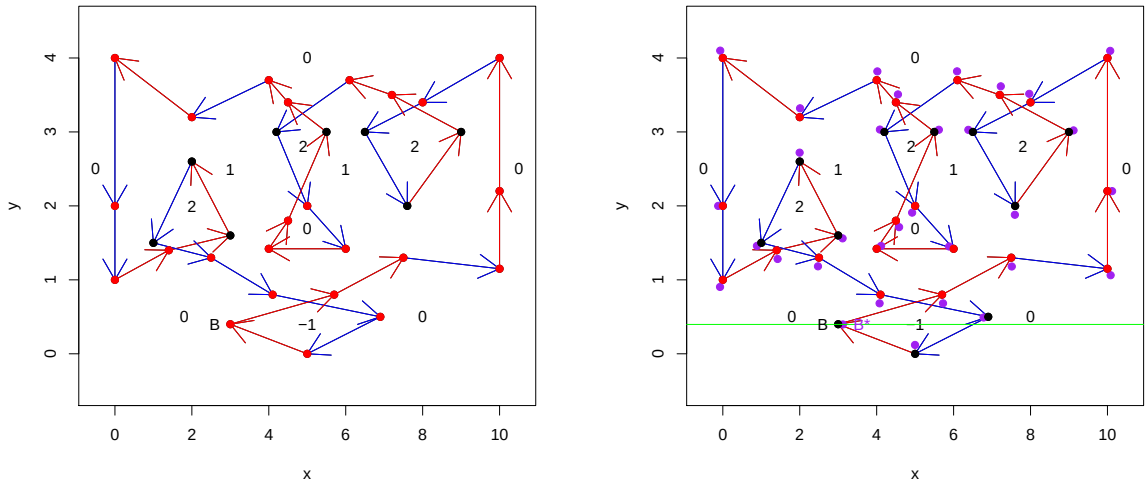


Figure 3: Both methods mark vertex A *inactive* but differ on the value of the turning number.

Although both methods find A to be *inactive*, the resulting turning number differs by one. Here, this discrepancy does not cause much alarm since we ultimately only want to know whether the vertex being evaluated is *active* or *inactive*, and both approaches do agree that A is *inactive*.

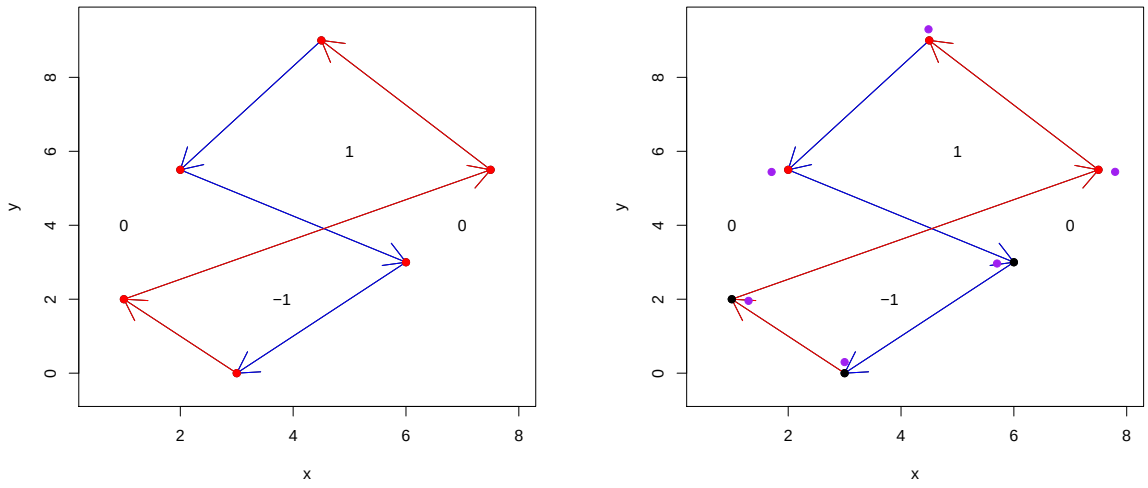
Example 2: Here we study a situation as shown in Figure 4. In (b), when vertex B is perturbed to vertex B^*, it is in a region with turning number -1, and hence by the *Scan Line* has turning number -1, and is *inactive*.
According to the Bryce-Richards algorithm, the turning number of the left hand region for B is zero. In this case, we do not proceed to find the turning number of the second region, and thus, mark B an *active* vertex—vastly different from the results of the *Scan Line*.

Example 3: Referring to Figure 5, we see that in (a), the Bryce-Richards algorithm marks all vertices as *active*, whereas using the *Scan Line* as in (b), only the top three vertices are *active*. However, a closer examination shows both methods are incorrect. This "figure 8" can arise from an elliptical fire burning whose rate of spread of the upper half is faster than that of the bottom half; thus, the fire is moving inward, so a loop occurs. This means, that the upper half should represent a region where the fire has crossed over itself; thus, the upper half of the "figure 8" is a region without fuel (since it is already burned), so those three vertices should be marked *inactive*.
Because these "figure 8" shapes can occur and we see a discrepancy in the current detection of *active*/*inactive* vertices, we aim to find another model that will accurately mark the vertices. Note that the algorithm *Prometheus* uses to distinguish active from inactive points does work in the majority of cases; in the instances where it does not work, however, the resulting fire fronts are highly inaccurate. The algorithms/methods proposed hereafter are intended to minimize these inaccuracies.



(a) (b)

Figure 4: In (a), the Richards and Bryce Approach marks B *active*, whereas in (b), the *Scan Line Approach* marks it *inactive*.



(a) (b)

Figure 5: In (a), the Bryce-Richards algorithm marks all vertices *active*, whereas in (b), the *Scan Line* marks only the top three vertices *active*.

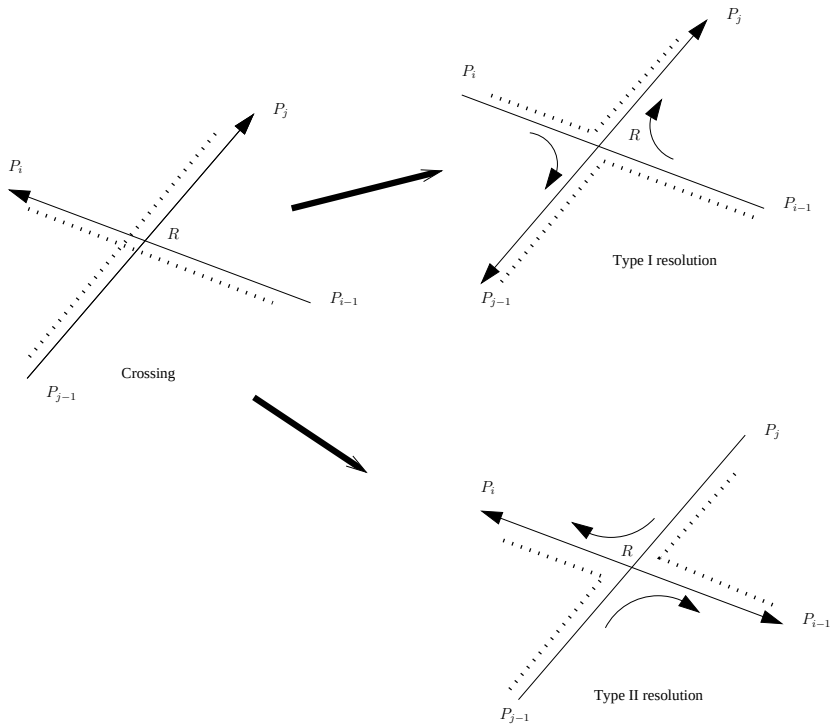


Figure 6: A crossing of the fire front with burned areas to the left.

2.2 Resolving Crossings – the Outer Hull Approach

The basic observation for this section is that every crossing of the fire front generates an ambiguity in the determination of some area as burned or unburned. Figure 6 makes the problem clear. In the figure, the crossing of two oriented segments from the front is shown on the left. Burned areas (to the left of the oriented segments) are indicated by dashed lines. The crossing determines (locally) four disconnected components. While two of the components are consistently labelled as burned or unburned, the other two necessarily have ambiguous labels. We wish to resolve this ambiguity by reversing the direction on some parts of the path and at the same time replace the crossing with a simple point of contact. There are two (non-isomorphic) ways to make this resolution, indicated by the Type I and Type II resolutions in the figure. The choice of which resolution to construct is determined by the incoming edge, on which it is assumed that the burned region is correctly determined. For example, if the first half of edge $P_{i-1} \to P_i$ is known to be correctly oriented, then we would chose the Type I resolution, exiting the crossing now at P_j with original orientation. The remaining two segments would be traversed with reverse orientation. To see concrete examples of how such crossings can arise in practice, look ahead to the figures starting with Figure 8.

Since both Type I and II resolutions change the orientation of some segments forming the original path, if we assume for the moment that the original curve was a single loop (a continuous image of the unit circle), possibly with crossings, after one crossing resolution, we are left with one, or possibly two loops which meet, but no longer cross at the point R. It is productive to think about this point R as being split into two copies, very close together so that the new paths are

actually topologically disjoint.

In order to describe the proposed algorithm, we will assume that we have been handed ONE tangled loop as the output from the front advance subroutine. The method we propose for delooping involves both untangling via crossing resolution and clipping to remove inactive segments. The output will be finitely many disjoint, simple, closed loops in the plane representing the active fire front. The extension of this algorithm to handle multiple (not necessarily disjoint) loops as input is straightforward.

First, identify some segment on the outer hull of the curve and label it's initial point as the base vertex. The assumption will be that the burned side of this segment is correctly determined laying 'to the left' as defined by the segment orientation.

Next, traversing the path from this initial segment and following the orientation, resolve crossings as they are encountered according to one of the two types. At each crossing this will have the effect of 'turning right' at the intersection point relative to the incoming orientation. After each crossing resolution, one will be left with one or more connected (but not necessarily simple) oriented paths, one of which contains the outer hull.

The process is iterated until the entire outer hull has been traversed by returning to the base vertex. Thus the outer hull is a simple, connected path, by construction.

Finally, one-by-one, crossings of any connected paths which do not lie on the outer hull are resolved. For each crossing, an 'incoming' edge is identified and Type I or II crossings are chosen so as not to change incoming directions. One is left with finitely many disjoint, simple closed paths, one of which is distinguished as the outer hull.

Clipping proceeds as follows:

All segments in the outer hull are retained.

For each path inside the outer hull which contains a segment whose orientation has changed from it's original orientation, we remove the whole path. The justification is that such a segment has been burned on both sides, and, by continuity, the same conclusion must be drawn for each segment on the path.

We now present the details. First, a vertex on the outer hull must be identified. We choose a value i_* so that $y_{i_*} = \min\{y_k \mid 0 \leq k \leq N\}$. The vertex $P_{i_*} = (x_{i_*}, y_{i_*})$ then represents a lower extreme point for the curve. Consider the orientation of the path $P_{i_*-1} \to P_{i_*} \to P_{i_*+1}$ If this is such that the burned area lies BELOW the segment (according to the left-hand rule) then we reverse the entire orientation of the curve at this point in the algorithm, since it is not physical for the lower extreme portion of the fire to have an exterior burned region. We need to do this in case this segment is part of a (non-physical) loop that was introduced from the previous front propagation step. Next we traverse the vertices in order from the base vertex P_{i_*} until the first crossing is encountered, say, on the $P_{j-1} \to P_j$ segment. Choose a crossing resolution (in this case, evidently of Type I) so as to not change the incoming orientation from P_{j-1} and reorder vertices according to the reordering algorithm described above. At this point the curve may split into two connected components or not.

It is useful to consider for a moment the effect of crossing resolution on the data structure, which we assume to be a vector of vertices $(P_1, P_2, \ldots, P_N)$ with $P_1 = P_N$. A crossing point R appears on two segments in this path and the resolutions consist of inserting two copies of the vertex R into the vector and reordering the components as in Figure 7. Note that one resolution disconnects the curve into two closed paths (loops), but the other leaves a single path. In either

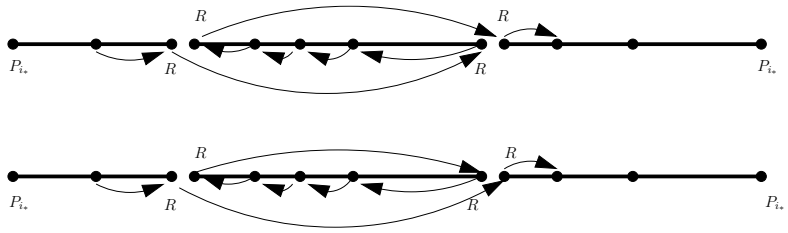


Figure 7: Resolution of crossing point – Data structure form. Crossing at vertex R

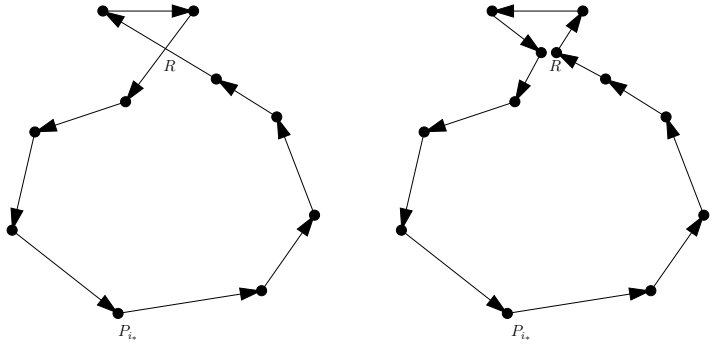


Figure 8: Merging an external loop

case we have:

Proposition 1 *In the case of crossing resolutions of Type II, the new path segment $P_{j-1} \to R \to P_{i-1}$ lies on a connected component of the base vertex P_{i_*} in the forward direction. Similarly, for resolutions of Type I.*

Proof. This is evident from Figure 7.

The uncrossing step is iterated until the path returns to the base vertex P_{i_*}. The resulting simple connected path is labelled the 'outer hull'.
Now return to the first resolved crossing R and (assuming resolution of Type I, the other case being similar) consider the path through $R \to P_{j-1}$. Continue to follow this path (which has had its order reversed) until one reaches again R, or until a crossing is reached. Crossings are to be resolved as encountered and with respect to the incoming order just as before. Proposition 1 shows that the vertex R must be attained since one cannot reconnect to the outer hull. This results in a second simple connected curve, disjoint from the outer hull.
Continue in this way until the entire curve has been resolved as a finite union of disjoint, simple connected curves.
Finally, except for the outer hull, delete any closed curve that contains an edge whose new orientation is reversed relative to its original orientation before the delooping algorithm started. The remaining curves (outer hull and interior loops) are the active fire front and can be fed into the next iteration of the fire-front propagation algorithm.

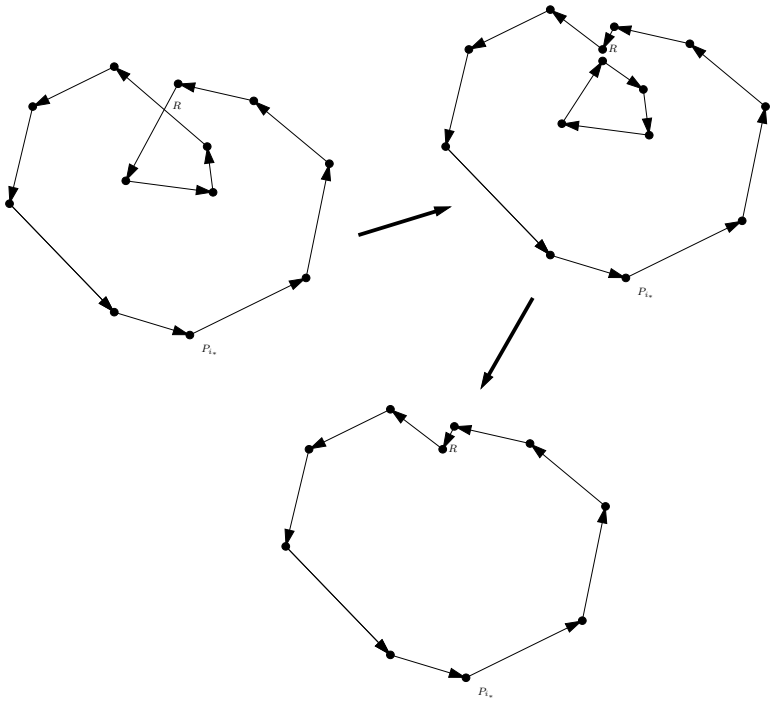


Figure 9: Clipping an internal loop

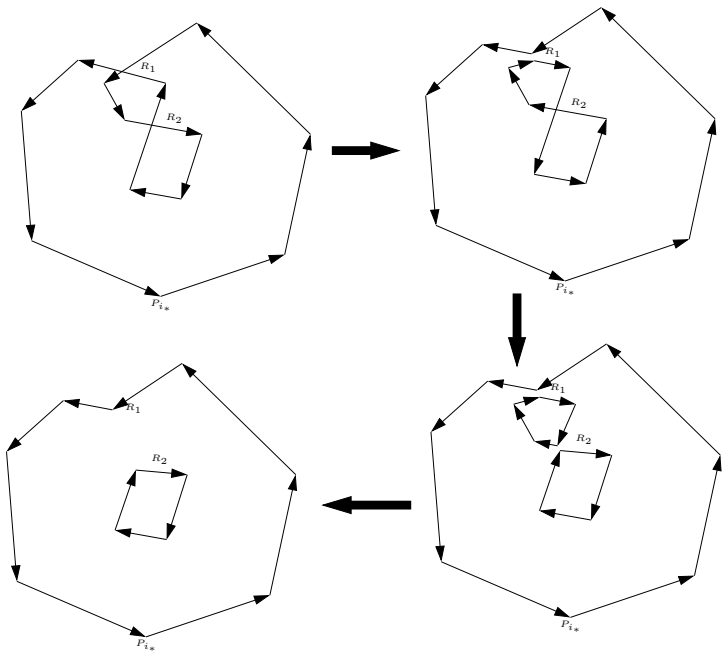


Figure 10: Horseshoe fire

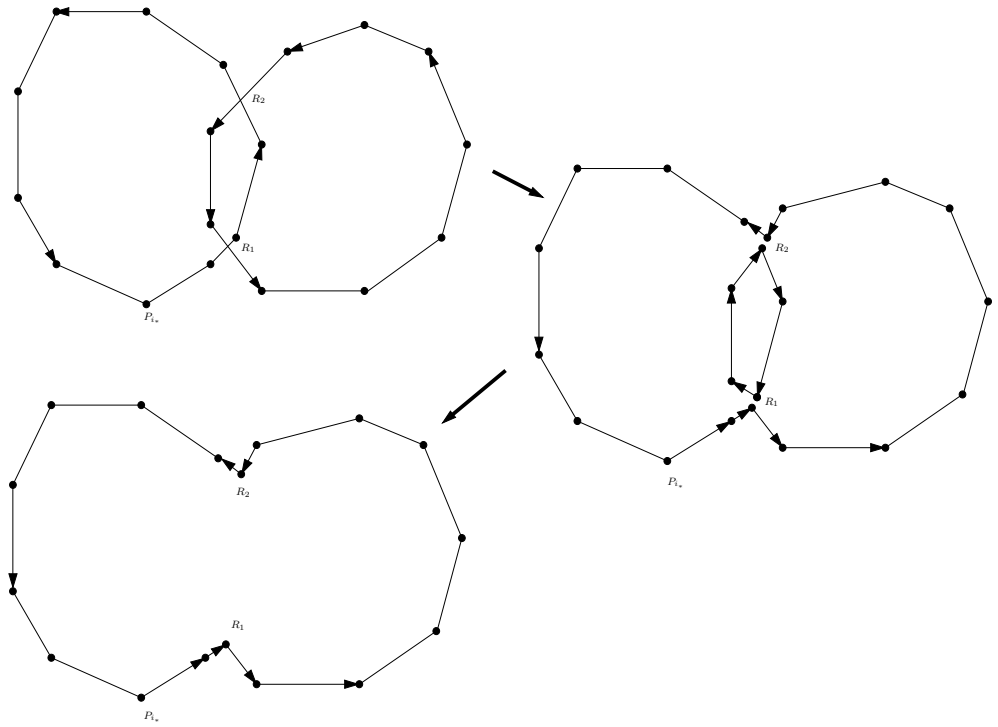


Figure 11: Merging of two simple fire fronts

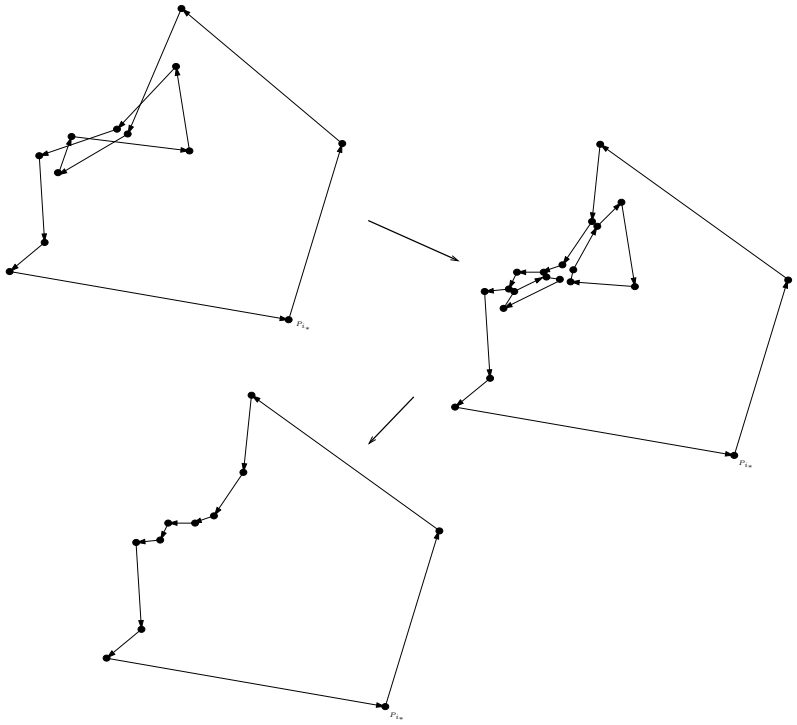


Figure 12: Resolution of the *Prometheus* nightmare example

2.3 Examples

1. **Merging an external loop.** In Figure 8 we see an external loop. The base vertex is chosen at P_{i_*} and there is a single crossing encountered at R. Resolving this crossing leaves the entire loop as the outer hull, with a pinched point at the crossing point R which, in the figure, we have split into two nearby vertices as discussed above. This example should be compared to the unsuccessful use of winding number techniques in Example 3 in Section 2.1.2 and Figure 5.

2. **Clipping an internal loop.** In Figure 9 we see an internal loop. Again, the base vertex is chosen at P_{i_*} and there is a single crossing encountered at R. Resolving this crossing disconnects the curve into two disjoint components. The interior loop is removed by the clipping rule as one (in fact all) segments end up with reversed orientation.

3. **Horseshoe fire.** In Figure 10 we see a fire that has grown two 'arms' which reconnect to enclose an unburned island. Base vertex P_{i_*} is easily identified and there are two crossings. R_1 is the first resolved crossing, resulting in a two disconnected curves, with orientation reversed on the inner figure eight. The outer hull has now been identified through return to the base P_{i_*}. Returning to R_1, which has now split into two nearby vertices in the figure, for clarity, we begin to traverse the figure eight loop with the new orientation. We next meet the crossing at R_2 which is resolved, breaking the figure eight into two simple loops. The orientation of the bottom loop is again reversed by the crossing resolution at R_2. Finally, the upper loop is clipped as (all of) its edges have reversed orientation, while the lower loop is retained as all of its edges have returned to the original orientation.

4. **Merging fires.** In Figure 11 we start with TWO non-disjoint loops. Base vertex P_{i_*} is identified as before. The first crossing encountered is the lower R_1 and the resolution is Type I. Note that the effect on the data structure at this point is to merge the two previously distinct paths (i.e., vectors not sharing any common points) into a single path connected at the crossing R_1. The resulting path continues to traverse the outer hull until the second crossing at R_2 is encountered. The resolution is again of Type I. Notice that the induced reorientation on the inner loop is consistent between the two resolutions – in both cases 'clockwise'. The path now returns to the base vertex and the outer hull is identified. The inner loop is now resolved easily, with no crossings, beginning and ending at R_1. The inner loop is clipped out and the figure-eight outer hull is all that remains.

5. **Prometheus nightmare.** This example was presented by *Prometheus* developers in [13] as a case where the winding number approach leads to incorrect results, but it is the kind of tangle "operators see time and time again". We omit details for application of the current algorithm, but show in Figure 12 the nightmare curve and its untangling according to the algorithm. Segments are now correctly identified as active or inactive and only the outer hull remains.

Remark. In Examples 3 and 4 where multiple crossings were encountered, it is important to observe that the local re-orientations induced by the sequential crossing resolutions are globally consistent. That is, orientations forced by an early uncrossing are consistent with orientations

required for later uncrossings on connected loops. So, we never end up with inconsistently directed polygons (two arrows coming together at a vertex). We have looked at many examples, some extremely complicated, and we always observe this feature. We conjecture that this is always the case, and that there must be a simple mathematical reason for it.

3 Smoothing

Smoothing the *Prometheus* parameters is another approach to minimizing the number of tangles introduced. In fact, smoothing has shown other advantages as well: it helps to reduce the data measurement error, and to aid in a residual-based bootstrap which allows for stochasticity in the model.

A study by T. Garcia, J. Braun, R. Bryce, and C. Tymstra was conducted to demonstrate the application of smoothing. Here we present a summary of their results. The data used in the study were the values of ROS, FROS, BROS, and RAZ from the Dogrib Fire that threatened Bearberry, Alberta in October 2001. Data values for the rates of spread were measured in metres/min, and the spread direction in azimuth (RAZ) was measured in degrees (for consistency, these units were converted to radians). The fire spread is evaluated on a portion of the terrain 4,950 m x 5,800 m, where the terrain is divided into a grid: 232 by 198 grid units. Each grid unit represents 25 square meters. Data values include those that represent non-fuel types such as rivers and lakes, and sloped areas in the terrain.

The smoothing methods used were `loess` and `locfit` both implemented in R. `loess` is a locally weighted polynomial regression, and `locfit` is another implementation of local regression that incorporates local likelihood techniques. Prior simulations showed that smoothing ROS and RAZ values had the most significant effects on the fire spread and only the smoothed values of these parameters were considered. After smoothing ROS and RAZ, their fitted values, $\widehat{ROS}$ and $\widehat{RAZ}$, were substituted into the ellipse parameters given in equations (12).

We used the `locfit` function, a non-robust form of local linear regression (see, e.g. [7]), to retain the important features of the terrain. This procedure uses a nearest-neighbor bandwidth to define a neighborhood of the fitting point. The neighborhood contains the points at which a weighted least squares line is to be estimated; we call the proportion of points relative to the entire set the smoothing parameter α.

Autocorrelation in the data makes it difficult to automatically select the smoothing parameter. Instead, we chose a small smoothing parameter to account for the regions of rapid change in ROS and RAZ (due to changes in terrain or fuel type), but not so small to incur lengthy computations. Ultimately, we chose $\alpha = 0.00025$ for $\widehat{ROS}$ which corresponds to a neighborhood of roughly 10 grid cells, and $\alpha = 0.01$ for $\widehat{RAZ}$. Also, the log scale was used to retain consistency of the ROS parameter. Rate of spread values equal to zero (those representing fire breaks) were removed from the data before smoothing was applied. Upon smoothing, the data were exponentiated, and the fire break observations were appended to the smoothed data set. This approach reduced the tendency for the smoothed values to decrease on approach to such fire breaks.

The effects of smoothing these inputs to the *Prometheus* model was exhibited through an R version of the *Prometheus* algorithm. One of the authors designed the program, which replicates the current *Prometheus* program but with one adjustment: in determining which vertices are

active or *inactive*, she used the Bryce-Richards algorithm, as opposed to the *Scan Line* used in *Prometheus*.
With α values set at 0.002, 0.01, 0.05, we tested three cases for smoothing at Δt=1, 2, 3 min:

1. Smoothing ROS values,
2. Smoothing RAZ values,
3. Smoothing ROS and RAZ values.

We chose Δt large so as to show the degree of tangles induced and the degree to which smoothing reduces these tangles.
We ran 27 simulations with the combinations produced by the different α, Δt, and 3 cases mentioned above. Our results indicate that smoothing the RAZ values often reduce the tangles present while still maintaining the features of the fire front. Occasionally, however, additional tangles are introduced. See Figure 13.
Smoothing the ROS values also reduces the number of knots induced, often more so than smoothing the RAZ values; see (b). Smoothing ROS and RAZ together also produces good results; see (d). While smoothing ROS and RAZ together does reduce the number of knots, in some instances, oversmoothing will lead to unrealistic patterns. Notice in (d), the fire is nearly a perfect elliptical shape; this is quite different from the original shape which has a jagged front, see (a).

3.1 Stochasticity via the Bootstrap

Our approach to incorporating stochasticity is a block bootstrap procedure. Blocks of residuals from the smoothing of ROS and RAZ are resampled and added back to the smoothed surfaces. We considered smoothing at Δt=3 and block sizes=10×10, 20×20, and 40×40, under the three cases:

1. Smoothing ROS values (using $\alpha = .00025$)
2. Smoothing RAZ values (using $\alpha = .01$)
3. Smoothing ROS and RAZ values

The results for block sizes of 20×20 and 40×40 differed only slightly from each other; hence Figure 14 only displays results for a block size of 40×40.
Referring to Figure 14, we observed that, as expected, incorporating randomness in the data values re-introduces knots to the fire front. Observe also that while the fire front shapes were changed due to randomness, the change was not extreme—the basic shape of the original fire front is present. Notice that when both ROS and RAZ are random, the number of knots present is more than when only one is considered random.
The results show that proceeding in this manner is a good start to introducing stochasticity and should be pursued. Further details of this study can be seen in the paper written by the authors T. Garcia, J. Braun, R. Bryce, and C. Tymstra. [11] and the recent Master's thesis by T. Garcia [12].

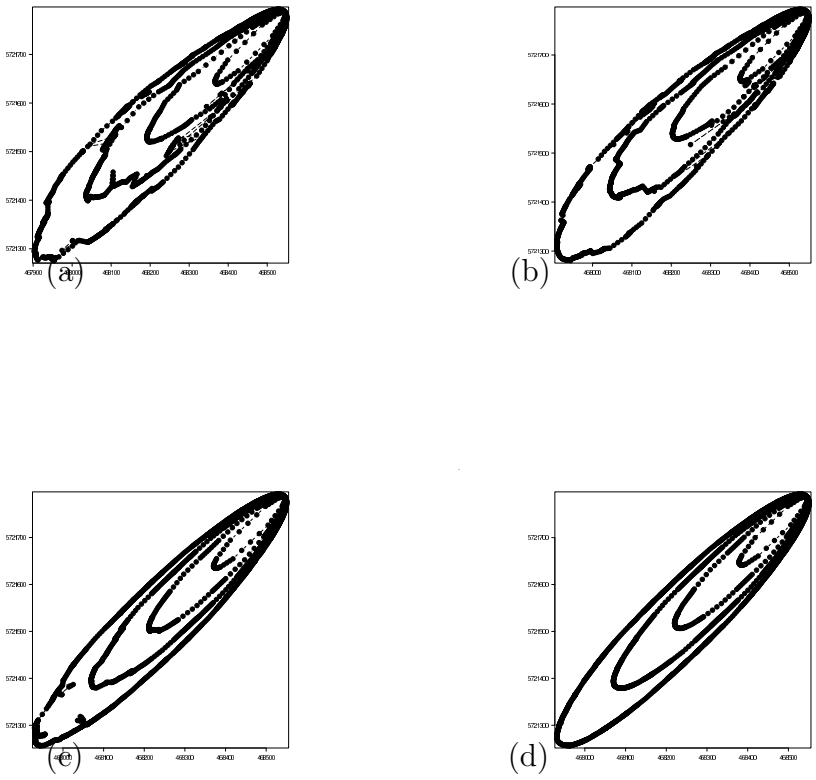


Figure 13: Output for *Prometheus* R version, $\Delta t = 3$, $\alpha = 0.05$ (a). Non-smoothed Data, (b). ROS smoothed only, (c). RAZ smoothed only, (d). ROS and RAZ smoothed.

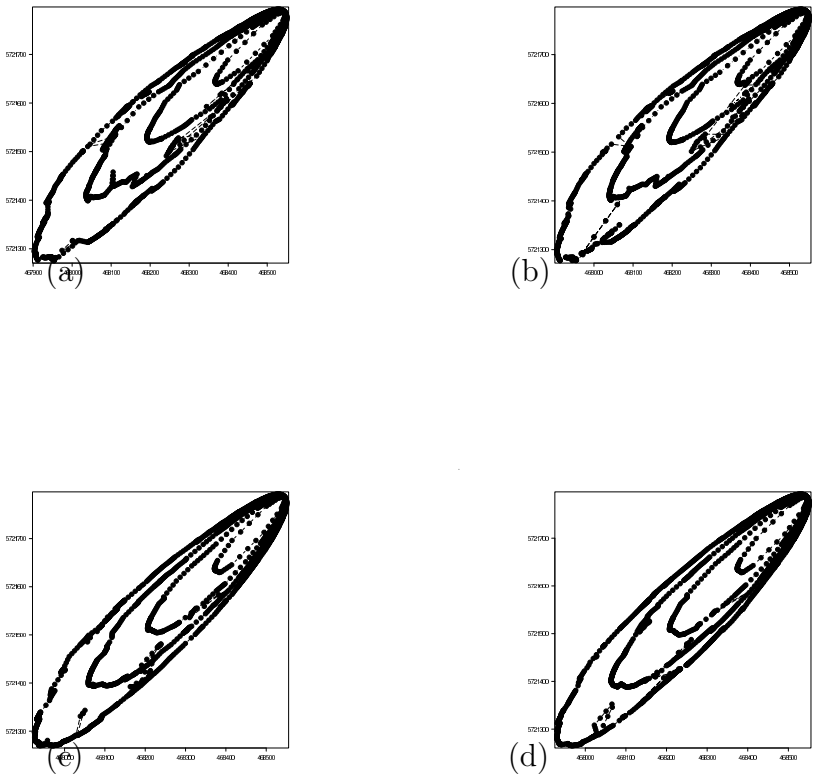


Figure 14: Output for *Prometheus* R version, $\Delta t = 3$, block size=40×40 (a). Non-random Data, (b). Random ROS data, (c). RAZ random, (d). ROS and RAZ random.

4 Redistribution of Vertices

In applications of the Lagrangian (marker) method, an initially uniform distribution of vertices along the fire front may not remain uniform; typically vertices begin to cluster in some areas along the curve, while in others their density may decrease substantially as large gaps between adjacent vertices appear. Clustering of vertices increases the possibility of tangles and crossovers, while, on the other hand, if the density of vertices is too low the course discretization can lead to inaccuracies in the computed fire front. In *Prometheus* this problem is addressed, to some extent, with *ad. hoc.* methods that combine vertices in clusters and introduce new vertices in gaps.

In this section we propose an automated approach to the redistribution problem based on *de Boor's Algorithm* [1] and *monitor functions*. A simple algorithm such as this, used to equidistribute vertices after a few front evolution steps, is likely a good and cheap investment for Prometheus developers.

4.1 Monitor Functions and de Boor's Algorithm

4.1.1 Monitor Function

To decide how many vertices are needed to resolve the front and where to place those vertices, we first need some measure of "how interesting a front segment is"; the idea being, the more interesting the front segment, the more vertices are required to resolve it. In the literature, there are several acronyms for such a measure, the most common being a *monitor function* or an *adaptation function*. A reasonable monitor function for the *Prometheus* fire growth problem should utilize (1) Arclength, (2) Curvature and (3) Rates of spread. One would like a cluster of grid points when the curvature is positive and large, and one would like fewer grid points when the curvature is negative and large. How one utilizes the rates of spread is probably dependent on the curvature.

To illustrate the idea of equidistribution and de Boor's algorithm, we first introduce the "arclength" monitor function,

$$\rho(u(x)) = \sqrt{1 + [u'(x)]^2},$$

where $y = u(x)$ is a function. In the example below, $\int_a^b \rho\, dx$ will result in the length of the graph $y = u(x)$. We will explain equidistribution and de Boor's algorithm at length in 1D before discussing a similarly motivated algorithm for the front (a "co-dimension one" object in 2D).

For a given monitor function, $\rho(u)$, (e.g the arclength monitor function), the idea behind equidistribution is to find a set of vertices $\{x_k\}$, $k = 0, \ldots, N$ such that

$$\int_{x_k-1}^{x_k} \rho(u(x))\, dx = \frac{\int_{x_1=a}^{x_N=b} \rho(u(x))\, dx}{N} \quad \forall k = 1 \ldots N, \quad a \le x \le b. \tag{13}$$

4.1.2 de Boor's Algorithm in 1D – Numerical Considerations

Suppose that the monitor function ρ is given only on some prescribed set of vertices, $\{x_k\}$, $k = 0, \ldots, N$ along the curve. Denote $\rho_k = \rho(x_k)$. The basic idea behind de Boor's algorithm may be adjusted by approximating $\rho(x)$ by a piecewise constant function based on the known values ρ_k,

$$\hat{\rho}(x) = \frac{1}{2}(\rho_{k-1} + \rho_k) \quad \text{for } x \in (x_{k-1}, x_k), \quad k = 1, \ldots, N, \tag{14}$$

and then find the equi-distributing vertices for $\hat{\rho}(x)$, this piecewise constant function. Denoting

$$I(x) = \int_a^x \hat{\rho}(\tilde{x})\, d\tilde{x},$$

and noticing that

$$I(x_k) = \sum_{j=1}^{k} (x_j - x_{j-1}) \frac{\rho_{j-1} + \rho_j}{2} \quad k = 1, \ldots, N,$$

the goal now is finding $\{y_k\}$, $k = 1, \ldots, N-1$, such that

$$I(y_k) = \frac{k}{N} I(b).$$

We pick N based on $I(b)$. The larger the total integral, the larger the number of vertices needed to resolve the front properly. To find $\{y_k\}$, $k = 1, \ldots, N-1$, let j be an integer such that

$$I(x_{j-1}) < \frac{k}{N} I(b) < I(x_j).$$

Since $I(x)$ is piecewise linear, $\{y_k\}$ can be calculated using

$$(y_k - x_{j-1}) \frac{\rho_{j-1} + \rho_j}{2} = \frac{k}{N} I(b) - I(x_{j-1}).$$

Note however that this algorithm only generates an equidistributed mesh to the piecewise polynomial defined in (14). To obtain a good approximation to the equidistributing mesh for a more general monitor function $\rho(x)$, an iterative scheme needs to be implemented. One can also implement a higher order interpolant, though the benefits for curve resolution vs. cost is unclear.

4.1.3 de Boors Algorithm for Fire Fronts

For equi-distributing on a fire front, we follow the same construction by approximating ρ to be piecewise constant on each segment connecting vertex x_i to vertex x_{i+1}. Denoting

$$I(s) = \int_a^s \hat{\rho}(\tilde{s})\, d\tilde{s},$$

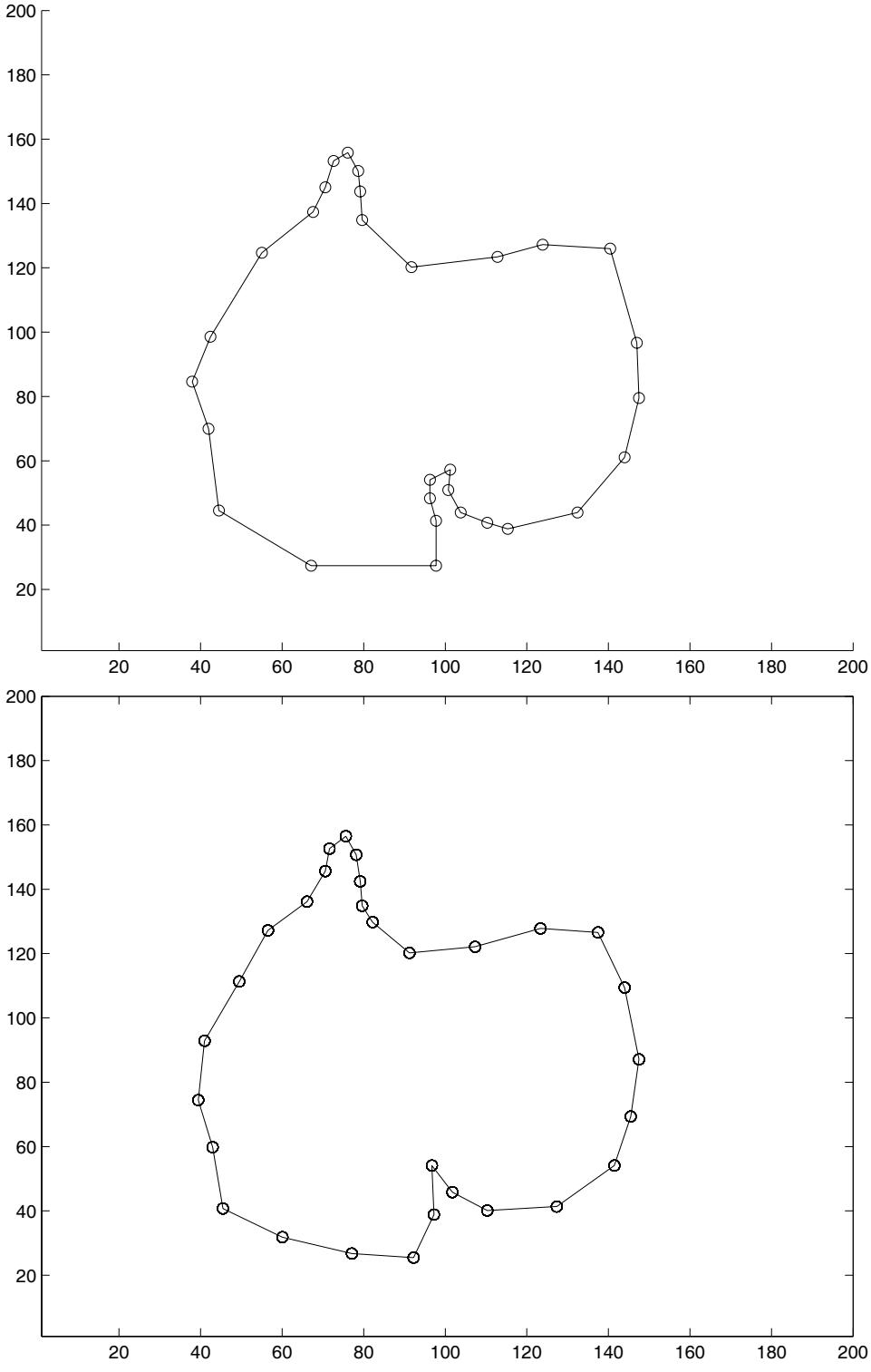


Figure 15: The top figure gives an example of propagated vertices, and the bottom figure shows equidistribution of vertices via a curvature based monitor function. Notice that there is some inherent smoothing due to the interpolation.

we assume that ρ is chosen so that

$$I(x_k) = \sum_{j=1}^{k} (d(x_j, x_{j-1}) + f(\kappa_j, v_j)) \quad k = 1, \ldots, N,$$

Here, $d(\cdot, \cdot)$ is the distance between the two vertices, κ_j is the (discrete) curvature at vertex x_j, v_j is some measure of the rate of spread at x_j, and $f(\kappa_j, v_j) \geq 0$ is part of the monitor function used to evaluate the "importance" of the front segment. In the current *Prometheus* implementation, $f \equiv 0$, for example.
Now, from the $I(x_k)$ obtain $\{y_k\}$, $k = 1, \ldots, N-1$, such that

$$I(y_k) = \frac{k}{N} I(b),$$

where we pick N again based on the magnitude of $I(b)$. To obtain $\{y_k\}$, $k = 1, \ldots, N-1$, we simply find j such that

$$I(x_{j-1}) < \frac{k}{N} I(b) < I(x_j).$$

and find y_k by interpolating between x_j and x_{j+1}. A short numerical simulation was run using a standard curvature based monitor function and the results recorded in Figure 15
We remark that in practice, the equidistribution process should be applied AFTER a delooping step (or not at all) as equidistribution applied to curves with crossings and other types of multiple points may lead to clustering of vertices, in fact.

5 Propagation by the Level Set Approach

The *Level Set* method is a state-of-the art numerical algorithm developed to handle interface propagation for a wide variety of applications ranging from the simulation of multiphase flows to image processing. Its underlying principle can be easily adapted for new problems as will be done here for the propagation of a forest fire. First, we summarize the general strategy. Then, we describe how issues specific to forest fire propagations can be tackled within the Level Set formulation: incorporating the elliptical growth model, selecting automatically the most appropriate time-step, merging distinct fire fronts into one front and including fire propagation barriers (for example, rivers). We will assess the performance of the Level Set approach by comparing the results with those obtained by the current *Prometheus* version for a benchmark test-case.

5.1 Basic Principle for the Level Set Method

The basic principle behind the method is to reformulate the problem of following the motion of the front in terms of the solution of a Hamilton-Jacobi partial differential equation (PDE) that is then solved (typically numerically) in the entire spatial domain. The method is intrinsically Eulerian, as opposed to the marker method which is an example of a Lagrangian approach.

The type of nonlinearity that characterizes Hamilton-Jacobi PDEs can lead to the formation of singularities in the solution (the appearance of such singularity typically coincides with marker paths crossings and tangles in the Lagrangian approach as previously observed).

Let us assume that initially, the front is described as the curve $\Gamma(0)$. The objective is to find the family of curves $\Gamma(t)$ which will represent the front at subsequent times t.
The *Level Set* method consists in embedding the front $\Gamma(t)$ at time t as the zero-level of a function $\phi(x, y, t)$ defined in the entire spatial domain. The function ϕ is such that it is positive on one side of the front (on the unburned side for example) and negative on the other side. The Level Set method approaches the problem of following the evolution of $\Gamma(t)$ in two steps. First, follow the evolution of the function $\phi(x, y, t)$, and then extract the zero-level of $\phi(x, y, t)$. Next, we show how to derive the evolution equation for $\phi(x, y, t)$ such that its zero-level evolves with the expected dynamic. For simplicity, we assume that at every point of the domain, one can compute $\vec{v}(x, y, t)$, the propagation velocity of the local iso-level of $\phi(x, y, t)$ so that it coincides with the fire front velocity on the zero-level. In the present application, this is done by considering each iso-level of ϕ as a virtual fire front and moving it with the local rate and direction of spread.
The curve $\Gamma(t)$ that represents the front at time t is given as the zero-level of $\phi(x, y, t)$:

$$\phi(x(t), y(t), t) = \phi(\vec{r}(t), t) = 0. \tag{15}$$

Differentiating with respect to t and using the chain rule, we get:

$$\phi_t + \nabla_{\vec{r}}\phi \cdot \frac{d\vec{r}(t)}{dt} = 0. \tag{16}$$

which is equivalent to

$$\phi_t + \vec{v} \cdot \nabla_{\vec{r}}\phi = 0. \tag{17}$$

In order to reduce clutter in our notation, from now on we will simply use ∇ to denote the spatial gradient, previously $\nabla_{\vec{r}}$.
For the forest fire application, $\vec{v} = \vec{v}(x, y, t, \vec{n})$, i.e. propagation velocity, in the most general case depends on space and time (for instance, local fuel properties, topography, weather data), as well as on the local orientation of the front as described by its unit normal $\vec{n}$ (selected as before to point towards the unburned side, in the direction of propagation of the fire front).

5.2 Level Set Formulation for the Elliptical Growth Model

First we recall that the elliptical growth model implies the following equation for the (continuous) front propagation vector $\vec{v}$ in (7) at the end of Section 1.2 :

$$\vec{v} = \begin{pmatrix} x_t \\ y_t \end{pmatrix} = \frac{A^T A\vec{n}}{\|A\vec{n}\|} + \vec{c} \tag{18}$$

where $A = \begin{pmatrix} b & 0 \\ 0 & a \end{pmatrix} \cdot R_\theta$, $\vec{c} = R_\theta^T(0, c)^T$ and R_θ is the rotation matrix from equations (2). The vector $\vec{n}$ was previously computed as $\vec{n} = (y_s, -x_s)$, with s a parametrization of the front

such that $\vec{n}$ is the correctly oriented unit normal to the curve. With the Level Set formulation proposed here, where $\phi > 0$ corresponds to the unburned side, the properly oriented unit normal can be obtained as $\vec{n} = \frac{\nabla\phi}{||\nabla\phi||}$. Substituting this and equation (18) into equation (17) gives the following

$$\phi_t + ||A\nabla\phi|| + \vec{c}\cdot\nabla\phi = 0 \tag{19}$$

The term $\vec{c}\cdot\nabla\phi$ has the form of an advection term by an externally specified flow $\vec{c}$ and $\|A\nabla\phi\|$ is the self-propagation term due to burning.

5.3 Numerical Implementation

5.3.1 Initialization of the Level Set Function

To start the simulation, one must specify initial conditions for $\phi(x, y, t = 0)$. It must be such that its zero-level coincides with the given curve $\Gamma(0)$. In the examples later in this report, we will use the standard approach, where $\phi(x, y, t = 0)$ is the signed distance function to $\Gamma(0)$, i.e. $\phi_0(x, y) = \phi(x, y, 0) = \sigma(x, y)d((x, y), \Gamma(0))$, where d is the shortest distance between (x, y) and $\Gamma(0)$ and

$$\sigma(P) = \begin{cases} -1, & \text{if } (x, y) \text{ is inside } \Gamma(0) \\ 0, & \text{if } (x, y) \text{ is on } \Gamma(0) \\ 1, & \text{if } (x, y) \text{ is outside } \Gamma(0). \end{cases} \tag{20}$$

For the forest fire application considered here, a typical initial configuration is that of a small circular fire around a specified ignition point. For such a case, one can use as initial value for the Level Set function ϕ the analytical solution for the signed distance function to that small ignition circle. For more general shapes, there are several numerical strategies to compute the signed distance function, see [6] and references therein for details. It must be noted however that, even if the initial data corresponds to the signed distance function to the initial front, the solution to the Level Set equation at later times will in most cases no longer correspond to the signed distance function to the front. Again, numerical strategies to regularly reinitialize the Level Set function to the signed distance function can be found in [6].

5.3.2 Discretization of the Level Set Equation

In the Level Set approach, one must first solve the unsteady PDE (17) in the entire spatial domain. A finite difference approach is used, where the spatial domain is meshed, with mesh sizes Δx, Δy in the x and y directions, respectively. The Level Set equation belongs to the class of Hamilton-Jacobi equations, whose type of nonlinearity can lead to discontinuities in the gradient of the solution, so great care must be taken when discretizing the PDE. We discuss here only the case when $\theta = 0$ in Eq(2) (wind aligned with the x-axis) and when $\vec{c} = 0$; it is straigthforward to generalize the scheme for other values of θ and also to include the standard advection term due to the velocity $\vec{c}$. With those restrictions, the Level Set equation is given by

$$\phi_t + (b^2\phi_x^2 + a^2\phi_y^2)^{1/2} = 0. \tag{21}$$

The proper discretization is a straightforward generalization from the eikonal equation (isotropic case, $a = b = 1$), see [6]. Let $\phi_{i,j}^n$ be the discrete approximation of $\phi(x_i, y_j, t^n)$, where x_i, y_j, t^n are the discrete space and time coordinates. Here is the scheme to compute $\phi_{i,j}^{n+1}$, the discrete solution at time $t^{n+1} = t^n + \Delta t$ and spatial coordinates (x_i, y_j), knowing the discrete solution ϕ^n at time t^n:

$$\phi_{i,j}^{n+1} = \phi_{i,j}^n - \Delta t(b^2\phi_{x;i,j}^2 + a^2\phi_{y;i,j}^2)^{1/2}$$

with

$$\phi_{x;i,j}^2 = max(min(\phi_x^+, 0)^2, max(\phi_x^-, 0)^2)$$

where

$$\phi_x^+ = \frac{\phi_{i+1,j}^n - \phi_{i,j}^n}{x_{i+1} - x_i}, \quad \phi_x^- = \frac{\phi_{i,j}^n - \phi_{i-1,j}^n}{x_i - x_{i-1}}.$$

and a similar algorithm for $\phi_{y;i,j}^2$.
This first-order accurate upwind scheme could be improved by using a more accurate approximation for the spatial and temporal derivatives. However, given the intrinsic modelling simplifications for the forest fire application considered here, it is expected that first order accuracy is adequate.

5.3.3 Selecting an Appropriate Time-Step

To insure that the numerical solution is stable, a standard analysis leads to a time-step restriction typically refered to as a "CFL" stability condition of the form :

$$\max_{domain} \{v_1, v_2\}\Delta t \leq \min\{\Delta x, \Delta y\} \tag{22}$$

with $\vec{v} = (v_1, v_2)$, the rate of spread of the front at any given point (x, y).

5.4 Front Merging

Unlike what happens with Lagrangian approaches such as the marker method, the topological changes that occur when two fronts merge do not require any particular treatment in the Level Set formulation. This is illustrated in the following figure. Two circular fronts are burning outward at a constant speed, as a result of which they eventually merge. The merging corresponds to a change of topology in terms of the fronts, but not in terms of the underlying Level Set function.

5.5 Barriers

Barriers are curves that the fire cannot cross, for example, a river sufficiently wide that one can assume the fire will not jump across. One simple way to take into account the presence of such

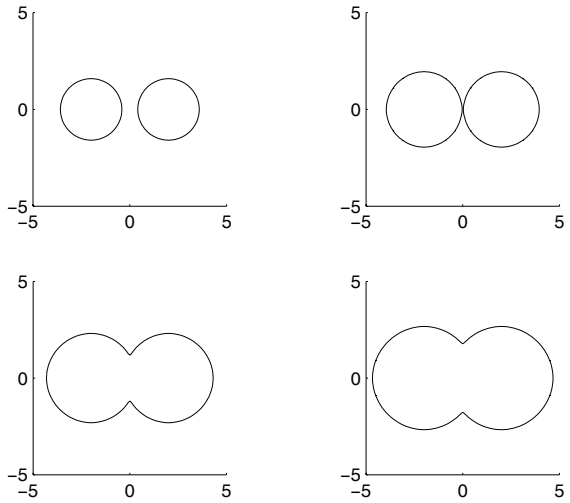


Figure 16: Merging of two circles

obstacles in the Level Set formulation is to impose that the propagation speed is strictly zero for all mesh nodes directly surrounding the barrier, as illustrated by an example in Figure 17. Figure 18 illustrates the good performance of the level-set solution as it propagates around the obstacle, with the two ends of the front eventually merging once they have reached around the the obstacle.

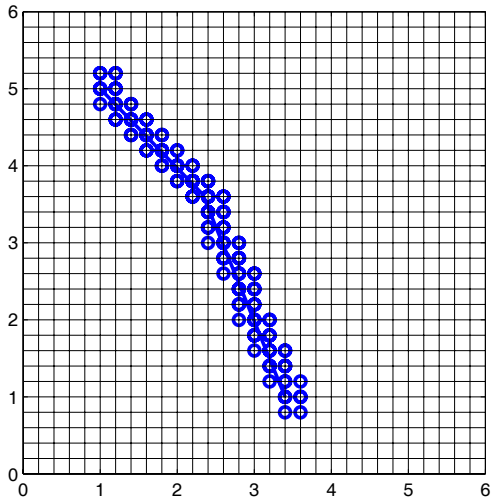


Figure 17: Nodes surrounding a barrier where the propagation speed is set to zero.

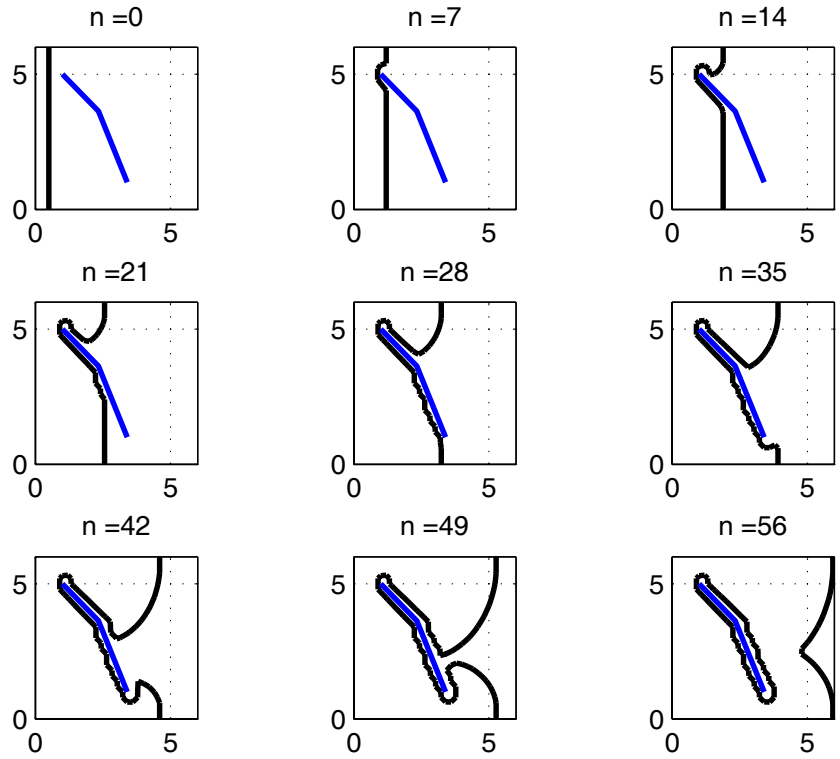


Figure 18: Example of propagation past the barrier in Figure 17 .

5.6 Simulation Results and Comparison

Here we compare the simulation of a fire propagation using the *Prometheus* algorithm, and using the Level Set approach. In both cases, the scale is $1 : 25m$ and the plots show the front at every hour. The total duration of the simulation is 6 hours. The vegetation is uniform except in one region (the dark rectangle) where the speed of propagation is higher. There is also a barrier, located in the upper rigth hand-side corner of the domain.

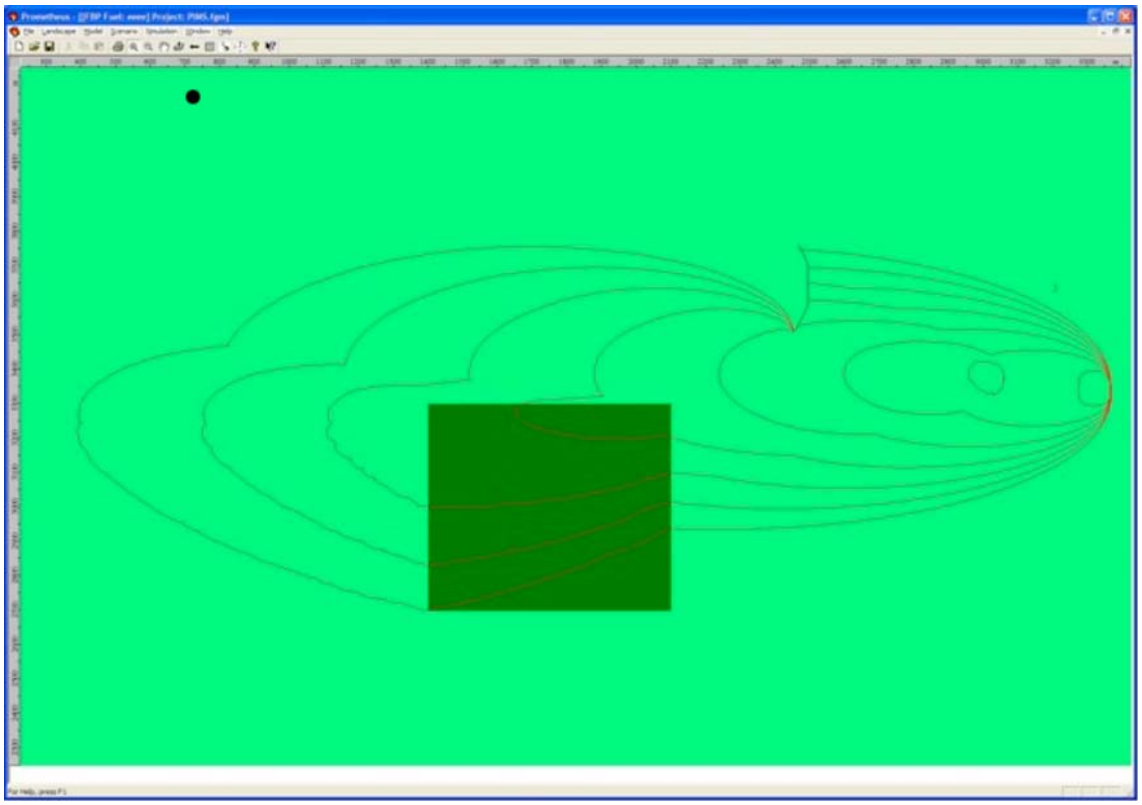


Figure 19: Simulation via *Prometheus* (screenshot)

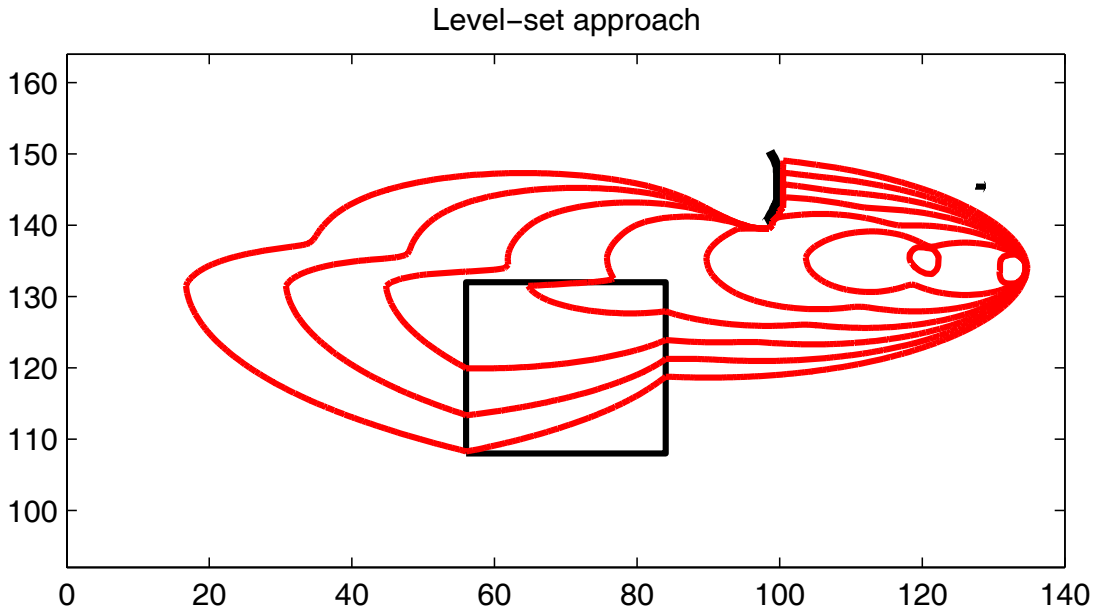


Figure 20: Simulation with the level-set method

There is overall a very good agreement between the reference results from *Prometheus* and the Level Set results. For instance, the position of the front after 6 hours in the vicinity of the obstacle is nearly identical for the two approaches. This is to be expected, as the underlying model is the same for both algorithms and that for this simple test-case, there is no significant tangling of the front.

6 Conclusions and Suggestions for Future Work

In Sethian's foundational book [6] a discussion of the Lagrangian approach to front propagation ends with a suggestion for three techniques for stabilizing an inherently unstable numerical scheme.

- Smoothing of the velocity function in order to allow for a reasonable time step.
- Redistribution of markers (vertices).
- Filtering to remove oscillations (noise) in the propagated front.

We have considered all three approaches in this study of the Prometheus model. We briefly summarize.
The first approach reported here concerns untangling and delooping in the Lagrangian formulation. *Prometheus* developers have coded up and tested the outer hull algorithm suggested in Section 2.2 and report both improved results (better identification of active/inactive segments of the front) and substantial speed improvements. Since the *Prometheus* scan-line algorithm represents large computational overhead in the *Prometheus* engine, this approach should be further developed and, if possible, investigated for performance on more complex tangles. It is possible to produce curves with high winding numbers where our *ad hoc* clipping rules fail to get it right – it would seem that the correct rules are still to be discovered. Finally, a rigorous proof of the global consistency of crossing resolutions seems mathematically feasible and a good first step to understanding this new approach.
Next we reported on smoothing of the parametric data defining the velocity field in equation (5) with the aim of regularizing the resulting field. Numerical results show that a great deal of control over the propagating front can be obtained this way – the challenge will be to find just the right level of smoothing so-as to retain the essential firefront features, and to reduce or eliminate the undesirable topological artifacts.

While the *Prometheus* code already allows for redistribution of vertices, any automatic or even more systematic way of doing this is of great interest to the developers. As we have shown, De Boor's algorithm shows promise on this aspect of the problem, but requires further testing.
The Level-Set approach shows great promise. Even during the short time-frame of the workshop, very impressive results on sample data were demonstrated. It is straightforward to import the propagation speed model data from traditional approaches into the Level-Set equations, and Level-Sets can handle features specific to forest fires, such as the presence of obstacles, and the merging of fronts. Testing of the full equations on a standard battery of model data would be a sensible next step.

Performance issues have not been discussed in this document, but there is a large literature on clever techniques to minimize the computational overhead with Level Sets. Such techniques could lead to better accuracy and more efficiency, opening the door to efficient large scale simulation of forest fires, especially for very heterogeneous landscapes and unsteady conditions as well as incorporating stochastic effects.
The model discussed here do not include wind or topographical features. This will be a natural next step to develop the model further.
The level set method is likely to require more memory than the corresponding marker method, since we will have to store information on the entire domain, or at least in a large enough region around the zero level set. This has, however, not been a limiting factor for the test cases presented here.
The level set method can as well be formulated for a non- uniform grid, say a finite element mesh. It is computationally more expensive as one loses the simplicity of a uniform mesh. Another approach is to use an "adaptive level set" function that clusters more grid points around breaks. While it is non trivial to code and "match" values on different resolutions, it preserves the advantages when evolving the level set function on a uniform grid.

References

[1] C. de Boor, Good Approximation by Splines with Variable Knots II, Springer Lecture Notes Series (Springer-Verlag, Berlin, 1973).

[2] G. Richards, A general mathematical framework for modelling 2 dimensional wildland fire spread, Int. J. Wildland Fire, **5**(2), (1994).

[3] G. Richards, An elliptical growth model of forest fire fronts and its numerical solution, Int. J. Num. Meth. Engineering, **30**, 1163-1179 (1990).

[4] G. Richards, Numerical simulation of forest fires, Int. J. Num. Meth. Engineering, **25**, 625-634 (1988).

[5] J. J. Helmson, A comparison of three-dimensional photolithography development methods. Ph.D. dissertation, EECS, University of California, Berkeley, 1994.

[6] J. Sethian, Level Set Methods and Fast Marching Methods, Cambridge University Press, 1999

[7] C. Loader 1999. *Local Regression and Likelihood* Springer: New York.

[8] R. Bryce and G. Richards, A computer algorithm for simulating spread of wildland fire perimeters for heterogeneous fuel and meteorological conditions, Int. J. Wildland Fire, **5** No. 2, 73-79 (1995).

[9] S. Osher and R. Fedkiw, Level Sets and dynamic implicit surfaces, Springer Applied Mathematics Series #153, 2003.

[10] R. Bryce and C. Tymstra. Vertex behavior and management. Working paper presented at the 10th IPSW problem solving workshop. Simon Fraser University. 2006.

[11] T. Garcia, J. Braun, R. Bryce, and C. Tymstra, Smoothing and bootstrapping the *Prometheus* fire spread model. Under review: *Environmetrics* Spring, 2007.

[12] T. Garcia. Improving the *Prometheus* model through parameter smoothing, Master's thesis (2006), University of Western Ontario.

[13] R. Bryce, Knots: A prometheus nightmare (2006). Unpublished working document.

[14] R Development Core Team, R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria (2006), ISBN 3-900051-07-0. `http://www.R-project.org`

[15] C. Loader, `locfit`: Local regression, likelihood and density estimation. R package version 1.5-3 (2006). `http://www.locfit.intro/`.