PROMETHEUS – Canada's Wildfire Growth Simulator

Jared Barber, Chris Bose, Anne Bourlioux,
John Braun, Eric Brunelle, Robert Bryce,
Tanya Garcia, Thomas Hillen, Clement Lam,
Ben Ong, Christian Poschl, Cordy Tymstra

DMS-861-IR

May 2007

Chapter 1

PROMETHEUS - Canada's Wildfire Growth Simulator

Jared Barber[1], Chris Bose[2], Anne Bourlioux[3], John Braun[4], Eric Brunelle[3], Tanya Garcia[5]
Thomas Hillen[6], Clement Lam[7], Ben Ong[8], Christian Pöschl[9]

Industrial Scientists: Robert Bryce and Cordy Tymstra

Report prepared by Chris Bose (cbose@math.uvic.ca) and Tanya Garcia
(tanya.garcia@unine.ch)

1.1 INTRODUCTION

PROMETHEUS is Canada's state-of-the-art wildfire growth modelling software program. Conceived in 1999 and developed over intervening years, PROMETHEUS now allows for accurate, fast, multi-day forecasts of moving fire fronts. Currently, the software is used in fire fighting situations, in fire risk analysis and in the design of 'fire-safe' communities and forests. The core of PROMETHEUS is an algorithm (the 'Engine') that calculates the evolution of the fire front using a marker method. The computational rules for moving the front are based on the well-accepted theoretical work of Richards [2], and Richards, Bryce [7]. The theory uses elliptical growth of a fire front on the 'microscopic' scale (eccentricity and orientation of axes based on local wind and slope conditions) and Hugyens' principle, treating the fire front as an infinite collection of microscopic, independent elliptical fires. This approach is very natural, and the implementation in PROMETHEUS gives good results in the field.

[1]Mathematics, The University of Arizona
[2]Mathematics and Statistics, University of Victoria
[3]Mathématiques et Statistique, Université de Montréal
[4]Statistical and Actuarial Science, University of Western Ontario
[5]Institut de Statistique, Université de Neuchâtel
[6]Mathematics, University of Alberta
[7]Computer Science, Concordia University
[8]Mathematics, Simon Fraser University
[9]Mathematics Institute, University of Innsbruk, Austria

1.2 PROBLEM DESCRIPTION

The industrial scientists who brought PROMETHEUS to the 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. Similar topological problems inherent with the Lagrangian approach are already mitigated by ad-hoc methods incorporated in the software but the results are not always satisfactory. Unified and/or rigorous methods for dealing with these issues would be helpful to the developers.

- 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.

- While PROMETHEUS already represents the state-of-the-art in Canadian wildfire modelling, the developers are interested in other computational approaches which would equal, or possibly exceed the performance of the marker method approach.

1.3 HUGYENS' PRINCIPLE – the Model Equations

In 1990, Gwynfor Richards at Brandon University proposed the basic model for fire propagation (see [7], [1], and [2]). Today the model is well-accepted from both theoretical and applied points of view, unlike earlier approaches using cellular automata as in [3]. 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. The 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 nonhomogeneity in the model parameters with respect to the spatial domain.

Assuming a locally smooth firefront and using Huygens' principle, the model states that the evolution of the next firefront is the envelope of small ellipses generated in a finite time interval by 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:

$$\begin{aligned} x(s,t) &= bt\cos s \\ y(s,t) &= at\sin s + ct \end{aligned} \tag{1.1}$$

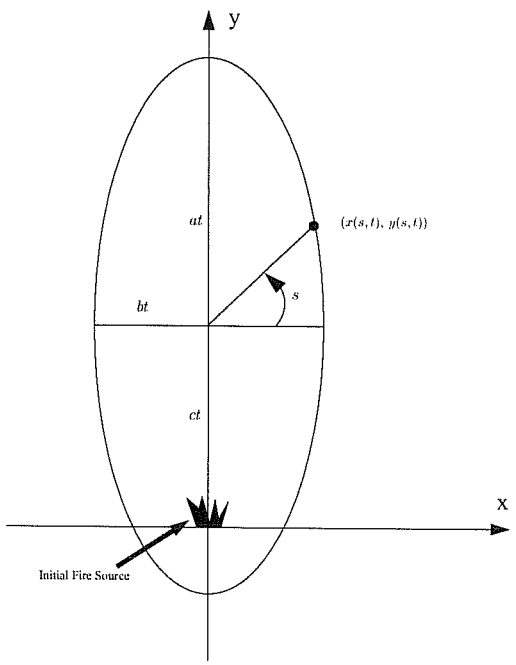


Figure 1.1: The elliptical shape of fire growth from a point at time t.

where t is time, s is a counter clockwise angle with respect to the horizontal line at $y = ct$ as in Figure 1.1 the point of ignition is assumed at the origin, and the wind direction is considered to be 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 values of ROS, FROS, and BROS are gridded parameters available from the Canadian Forest Fire Behaviour Prediction System (CFFBP). They are dependent on fuel type, temperature and moisture content, for example. Hence, the parameters a, b, c are spatially dependent: $a = a(x, y) \ldots$ and so on. In fact, for some long range models we should allow time dependency as well, but, for simplicity, in this report we will be content with just spatial dependence. One other parameter RAZ $= \theta$ is the effective wind direction, another gridded parameter that is taken as a model input. In our simplified elliptic model then, RAZ $= 0$.

Hugyens' principle views the active fire front at time t as an infinite collection of ignition points, each of which evolves independently according to Equation 1.1 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.

Here are the details.

Assume we can represent the coordinates of the fire front at time t by $(x(s,t), y(s,t))$, where $0 \leq s \leq S$. Notice, s mentioned above is an angle, whereas from this point forward it will be more convenient to assume s is arc length (from some distinguished point), and S the length of the entire curve. Then, the small elliptical fire generated by the point $(x(s,t), y(s,t))$ propagates at an angle $RAZ = \theta$ and defined by equation (1.1) 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 finite dt; and, taking the limit as dt approaches zero, we obtain the time derivatives $x_t(s,t)$ and $y_t(s,t)$, the solution we want.

We first calculate $(x(s,t+dt), y(s,t+dt))$ by transforming the ellipses into circles via a linear transformation T. One can then easily calculate the equations of the envelopes of circles by a limiting process in ds. Then, by applying the inverse of the linear transformation, one obtains $(x(s,t+dt), y(s,t+dt))$.

In more detail, 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

$$\Gamma = \begin{pmatrix} a/b & 0 \\ 0 & 1 \end{pmatrix} \tag{1.2}$$
$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

we have

$$T = \Gamma \cdot R_\theta. \tag{1.3}$$

Applying T to the elliptical coordinates $(x(s,t+dt), y(s,t+dt))$, we obtain the circular coordinates:

$$\begin{aligned} X &= [a(s,t)/b(s,t)](x\cos\theta - y\sin\theta) \\ Y &= x\sin\theta + y\cos\theta. \end{aligned} \tag{1.4}$$

To find the coordinates $(X(s,t+dt), Y(s,t+dt))$, proceed as follows: 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:

$$\begin{aligned} X(s,t+dt) &= X(s,t) + \frac{dta(-X_s dt\cdot a_s + (dt\cdot c_s + Y_s)((dt\cdot c_s+Y_s)^2 + X_s^2 - dt^2 a_s^2)^{1/2})}{((dt\cdot c_s+Y_s)^2+X_s^2)} \\ Y(s,t+dt) &= Y(s,t) + \frac{dt\cdot a(-((dt\cdot c_s+Y_s)^2+X_s^2-dt^2a_s^2)^{1/2}X_s - (dt\cdot c_s+Y_s)^2 dt\cdot a_s)}{((dt\cdot c_s+Y_s)^2+X_s^2)} + cdt \end{aligned} \tag{1.5}$$

These equations define the envelope of the circles formed in finite time dt.

Taking the inverse transformation, T^{-1}, and letting dt tend to zero, we obtain the differential equations modelling how the fire front, formed by the envelope of infinitesimally small ellipses, evolves in time:

$$\begin{aligned} x_t(s,t) &= \frac{b^2\cos\theta(x_s\sin\theta + y_s\cos\theta) - a^2\sin\theta(x_s\cos\theta - y_s\sin\theta)}{\sqrt{a^2(x_s\cos\theta - y_s\sin\theta)^2 + b^2(x_s\sin\theta + y_s\cos\theta)^2}} + c\sin\theta \\ y_t(s,t) &= \frac{-b^2\sin\theta(x_s\sin\theta + y_s\cos\theta) - a^2\cos\theta(x_s\cos\theta - y_s\sin\theta)}{\sqrt{a^2(x_s\cos\theta - y_s\sin\theta)^2 + b^2(x_s\sin\theta + y_s\cos\theta)^2}} + c\cos\theta \end{aligned} \tag{1.6}$$

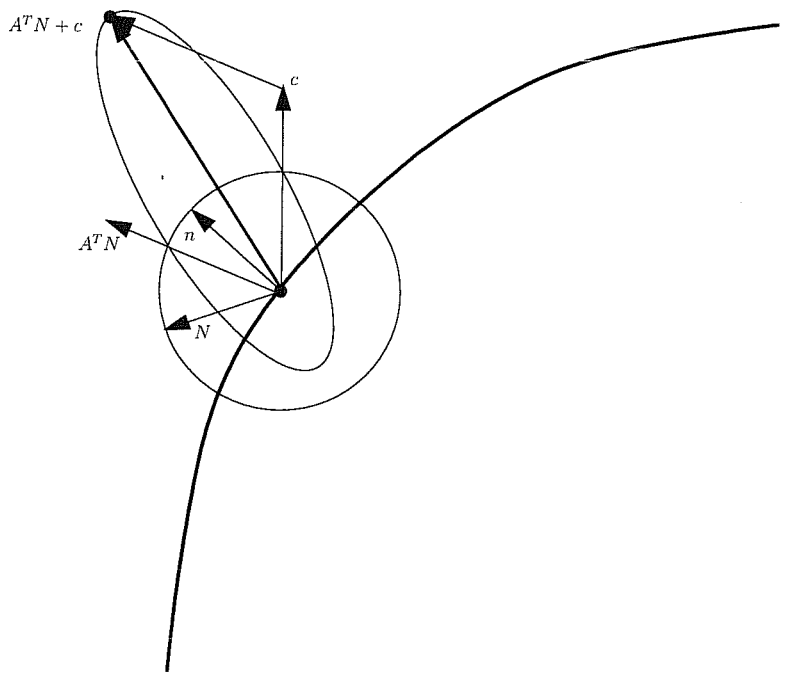


Figure 1.2: The graphical view of the reduced mathematical formulation for the differential spread equations.

subject to the initial conditions

$$\begin{aligned} x(s,0) &= x_0(s) \\ y(s,0) &= y_0(s). \end{aligned} \tag{1.7}$$

Note that while these equations are consistent with the presentation given in [9], there is an exchange of the parameters a and b when compared to the original paper of Richards [2].

Observe that the equations in (1.6) can be succinctly rewritten as:

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \frac{A^T A\overrightarrow{n}}{\|A\overrightarrow{n}\|} + \overrightarrow{c} \tag{1.8}$$

where $A = \begin{pmatrix} b & 0 \\ 0 & a \end{pmatrix} \cdot R_\theta$, $\overrightarrow{n} = (y_s, -x_s)$, $\overrightarrow{c} = R_\theta^T(0,c)^T$. Notice R_θ is an orthogonal matrix, so $R_\theta^{-1} = R_\theta^T$.

We can see this visually in Figure 1.2. In this figure, N denotes $\frac{A\overrightarrow{n}}{\|A\overrightarrow{n}\|}$.

1.4 THE PROMETHEUS ENGINE – the Marker Method for Front Propagation

While the equations in (1.6) are analytically intractable, they can be solved numerically, as done in the PROMETHEUS model.

Discretize the curve $(x(s,t), y(s,t))$ into n points such that

$$(x_{i,j}, y_{i,j}) = (x(ids, jdt), y(ids, jdt)) \tag{1.9}$$

where $ds = S/n$, and ds and dt are the parameter and time step sizes, respectively, and $(x_{0,j}, y_{0,j}) = (x_{n,j}, y_{n,j})$. Then, using a centered-difference in s and forward difference in time, we can solve for $(x_{i,j+1}, y_{i,j+1})$. To do this, first observe that the equations in (1.6) are functions of s and t (recall, the parameters a, b, and c are functions of s and t), so we can rewrite them as:

$$\begin{aligned} x_t &= F(s,t,x_s,y_s) \\ y_t &= G(s,t,x_s,y_s). \end{aligned} \tag{1.10}$$

Using the central difference approximations for derivatives x_s and y_s, we have the following numerical method; letting,

$$dx_{i,j} = dtF(ids, jdt, (x_{i+1,j} - x_{i-1,j})/2ds, (y_{i+1,j} - y_{i-1,j})/2ds) \tag{1.11}$$

$$dy_{i,j} = dtG(ids, jdt, (x_{i+1,j} - x_{i-1,j})/2ds, (y_{i+1,j} - y_{i-1,j})/2ds) \tag{1.12}$$

we have:

$$\begin{aligned} x_{i,j+1} &= x_{i,j} + dx_{i,j} \\ y_{i,j+1} &= y_{i,j} + dy_{i,j} \end{aligned} \tag{1.13}$$

The PROMETHEUS model implements this finite difference solution. First, the terrain is divided into a grid, where each grid-cell is identified by fuel type and rates of spread. The initial fire is formed by a circle of 32 vertices ordered counter clockwise, with its radius 1/5th the grid-cell size. At iteration j, the solution for the $j+1$ iteration is done as follows. Define the following notation:

Δt is the duration of the time step applied to a given simulation. Currently, Δt is constant for all time steps.

$\overrightarrow{P_j}$ is the vector of points or vertices defining the fire front at time iteration j, equivalently time $j\Delta t$.

P_j^i is the i^{th} vertex (i.e., the i^{th} component of $\overrightarrow{P_j}$) on the fire perimeter at time iteration j.

$P_{x_j}^i$, $P_{y_j}^i$, $P_{z_j}^i$ are the components of P_j^i. Note $P_{z_j}^i$ is zero in the two-dimensional PROMETHEUS representation for all i and j.

$\Delta P_j^i = P_{j+1}^i - P_j^i$ is the change in location of point i from current time step j to $j+1$.

ROS_j^i, $FROS_j^i$, $BROS_j^i$, RAZ_j^i are the CFFBP outputs ROS (forward rate of spread), FROS (flank rate of spread), BROS (back rate of spread), RAZ (spread direction in azimuth, θ, clockwise, from positive y-axis), respectively, for point P_j^i.

We then find the locations for the $j+1$ iteration by solving the following equations[10]:

$$\Delta P_{x_j}^i = \Delta t \frac{b^2 \cos\theta(x_s \sin\theta + y_s \cos\theta) - a^2 \sin\theta(x_s \cos\theta - y_s \sin\theta)}{\sqrt{a^2(x_s \cos\theta - y_s \sin\theta)^2 + b^2(x_s \sin\theta + y_s \cos\theta)^2}} + c\Delta t \sin\theta \tag{1.14}$$

$$\Delta P_{y_j}^i = \Delta t \frac{-b^2 \sin\theta(x_s \sin\theta + y_s \cos\theta) - a^2 \cos\theta(x_s \cos\theta - y_s \sin\theta)}{\sqrt{a^2(x_s \cos\theta - y_s \sin\theta)^2 + b^2(x_s \sin\theta + y_s \cos\theta)^2}} + c\Delta t \cos\theta$$

where

$$\begin{aligned} a &= \frac{ROS_j^i + BROS_j^i}{2} \\ b &= FROS_j^i \\ c &= \frac{ROS_j^i - BROS_j^i}{2} \\ \theta &= RAZ_j^i \\ 2x_s ds &= P_{x_j}^{i+1} - P_{x_j}^{i-1} \\ 2y_s ds &= P_{y_j}^{i+1} - P_{y_j}^{i-1} \end{aligned} \tag{1.15}$$

Notice these equations are consistent with those of (1.6). For example, solving for ROS above, for fixed i and j, yields forward rate ROS_j^i as $a+c$, and flank rate FROS_j^i as b.

1.5 TANGLES AND OTHER TOPOLOGICAL CONSIDERATIONS – how not to get burned twice

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 [5] gives a good overview of the issues involved. See also [4] 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 have considered 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.

[10] We again remind the reader that there is an inconsistency in the literature regarding the use of a and b – we follow the convention from [9], whereas in the reference [2] the roles of a and b are switched.

First, we discuss in some detail the current approach used by PROMETHEUS: a winding number calculation followed by heuristics for untangling. It was discovered during the course of the workshop that the two numerical methods (*scan line* vs. the *Bryce-Richards algorithm*), previously thought to be equivalent, can produce inconsistent results even for fairly simple fronts. Worse, both methods can fail to correctly determine the correct state of a vertex on a knotted firefront.

Next, we discuss three approaches to avoid the production of knots and crossovers in the first place. Parameter smoothing (detailed in Section 1.5.3) is based on the idea that crosssovers arise from discontinuities in the discretized spatial data at the front propagation step. Another natural idea is to decrease the time step Δt when topological instabilities are encountered, thereby avoiding or at least minimizing crossovers (Section 1.5.6). Some sort of (*regridding* or *regularization*) of the vertex set on the front seems necessary to take advantage of this adapted timestep approach. In fact, the present PROMETHEUS has a number of heuristic rules for adding new vertices to the evolving front. Further suggestions are presented in Section 1.5.6. A more sophisticated approach to regularization via *monitor functions* and *de Boor's Algorithm* is presented in Section 1.5.7. Finally, in Section 1.5.8 we discuss a completely new idea for identifying active segments on the fire front, and for removing inactive ones based on sequential resolution of crossings and knowledge of an *a priori* hierarchy of fire front segments.

1.5.1 The existing approach to Knots and Crossovers

As discussed in section 1.4, 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 (1.14) and (1.15).

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.

1.5.2 Determining Active and Inactive Vertices

To date, 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" [7]. 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:

1. Bryce-Richards algorithm [7].

2. Scan Line

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.

Methods to determine the Turning Number of a Vertex

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.

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.

Determination of the turning number of the left hand region

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.

Determination of 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 labelled 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.*

Scan Line

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.

Discrepancy between the two methods

After a discussion with Robert Bryce, 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 examples illustrate the discrepancy.

Description of Figures

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 [7]; this example (see Figure 1.4) instigated the discussion of the discrepancies observed. All the figures drawn here are the result of an R program [13] 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.

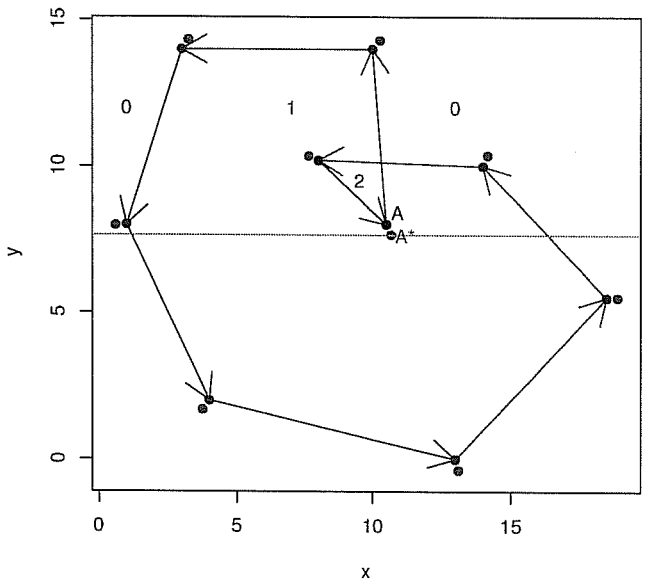


Figure 1.3: Both methods mark vertex A *inactive* but differ on the value of the turning number.

Example 1: An Inactive Vertex becomes More Inactive

Refer to Figure 1.3. This figure was designed as a way of understanding the *Scan Line*; 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.*

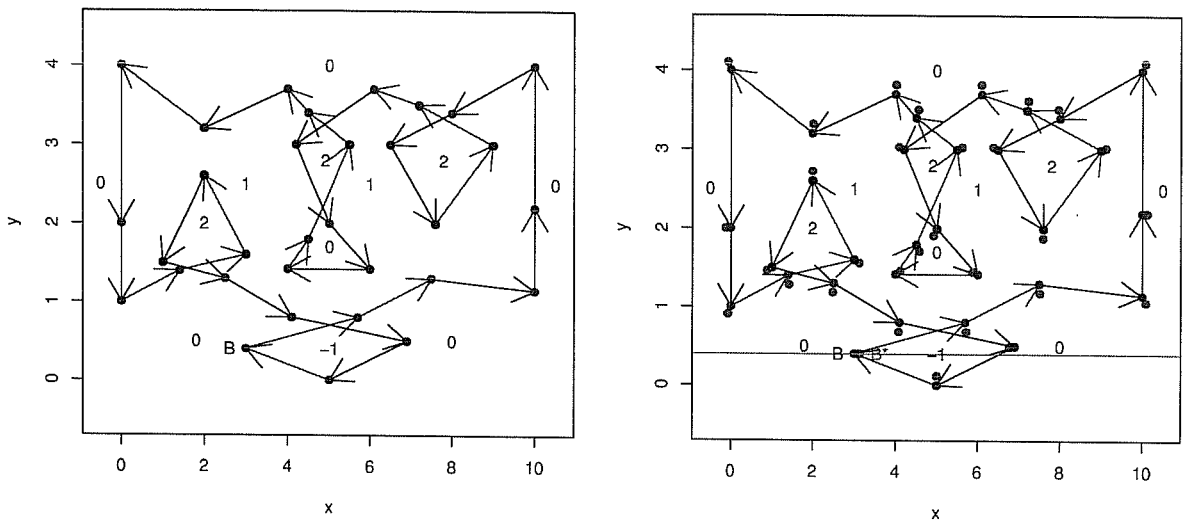
However, our results differ when we implement the Bryce-Richards algorithm. Following the algorithm, we first determine the turning number of the left hand region of A. When doing so, we find that the left hand region has turning number 1. Because this is non-zero, we must find the turning number of 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.*

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: Inactive Vertex becomes Active

Refer to Figure 1.4. Here, 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.*



(a) (b)

Figure 1.4: In (a), the Richards and Bryce Approach marks B *active*, whereas in (b), the *Scan Line Approach* marks it *inactive*.

Example 3: Both Methods Incorrect

Referring to Figure 1.5, 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 heuristic 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.

1.5.3 Application of Smoothing and Bootstrapping

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,

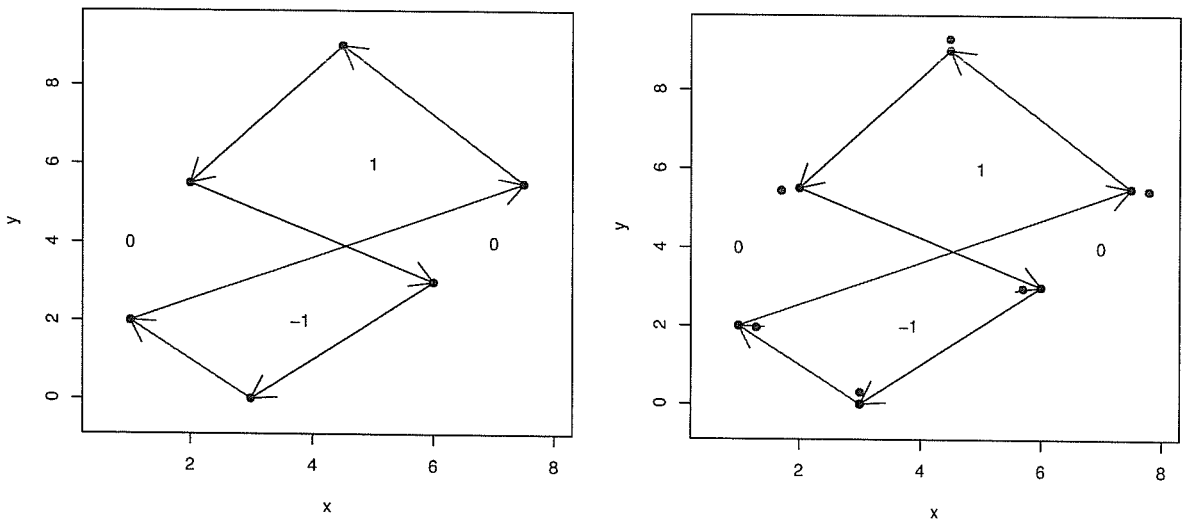


Figure 1.5: In (a), the Bryce-Richards algorithm marks all vertices *active*, whereas in (b), the *Scan Line* marks only the top three vertices *active*.

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 firespread 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 firespread 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 (1.15).

We used the `locfit` function, a non-robust form of local linear regression (see, e.g. [6]), 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 fueltype), 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.

1.5.4 Smoothing Reduces Tangles

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 1.6.

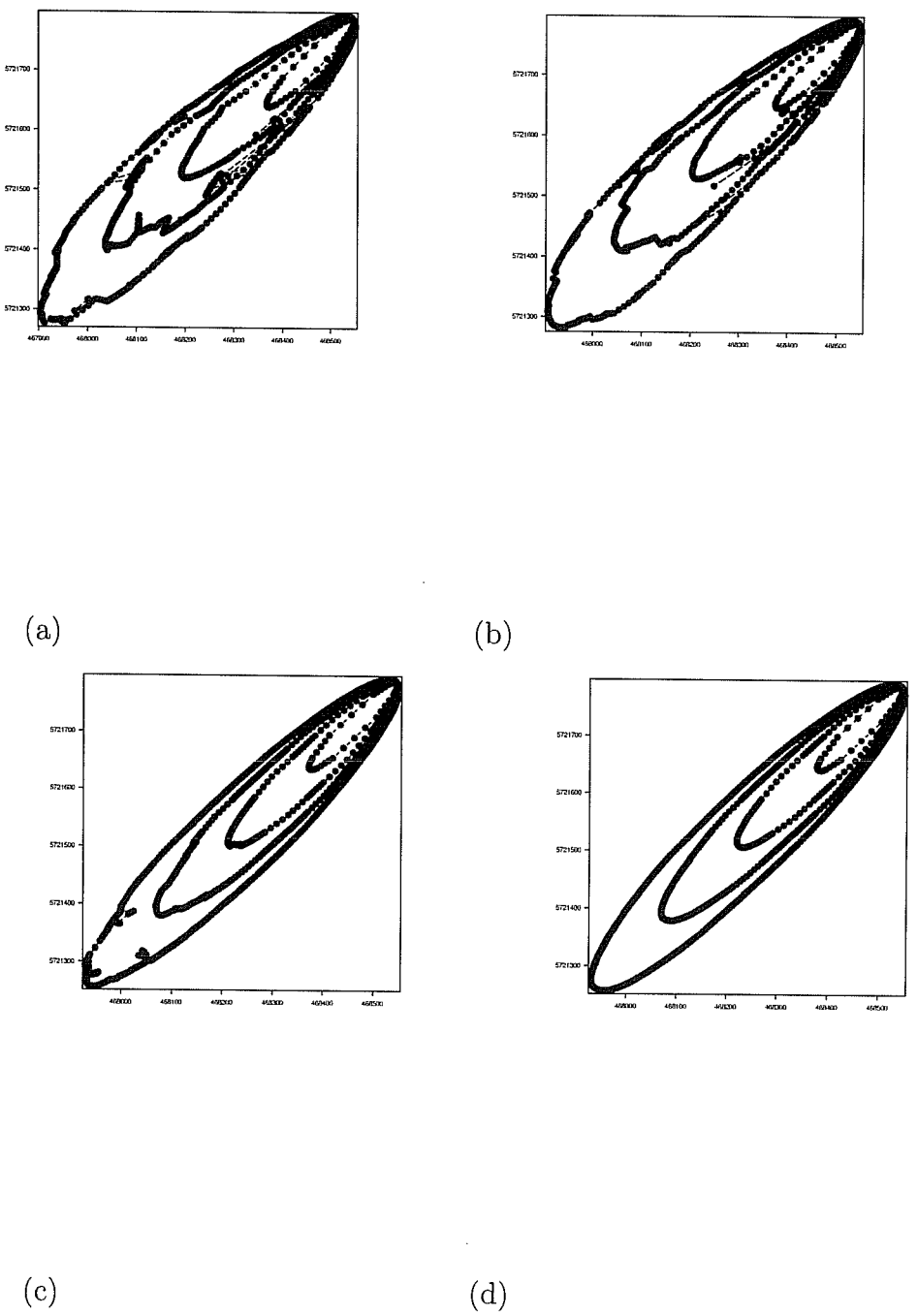
Smoothing the ROS values also reduces the number of knots induced, often moreso than smoothing the RAZ values; see (b). Smoothing ROS and RAZ together also produces good results; see (d). While smoothing ROS and RAZ together does reduce the number of knots, in some instances, oversmoothing will lead 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).

1.5.5 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



(a) (b)

(c) (d)

Figure 1.6: 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.

The results for block sizes of 20×20 and 40×40 differed only slightly from each other; hence Figure 1.7 only displays results for a block size of 40×40.

Referring to Figure 1.7, 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. [10] and the recent Master's thesis by T. Garcia [11].

1.5.6 Adapted Timestep and Regularization – I

Ideas on variable time step calculation

Given:

1. Multiple simple closed fronts represented by circular lists of vertices.
2. A fire propagation model.
3. A grid resolution, denoted by l_g (nominal units: meters).
4. A time step t_s (in seconds).

Part 1: Rules for removing vertices too close to one another.

Methodology:

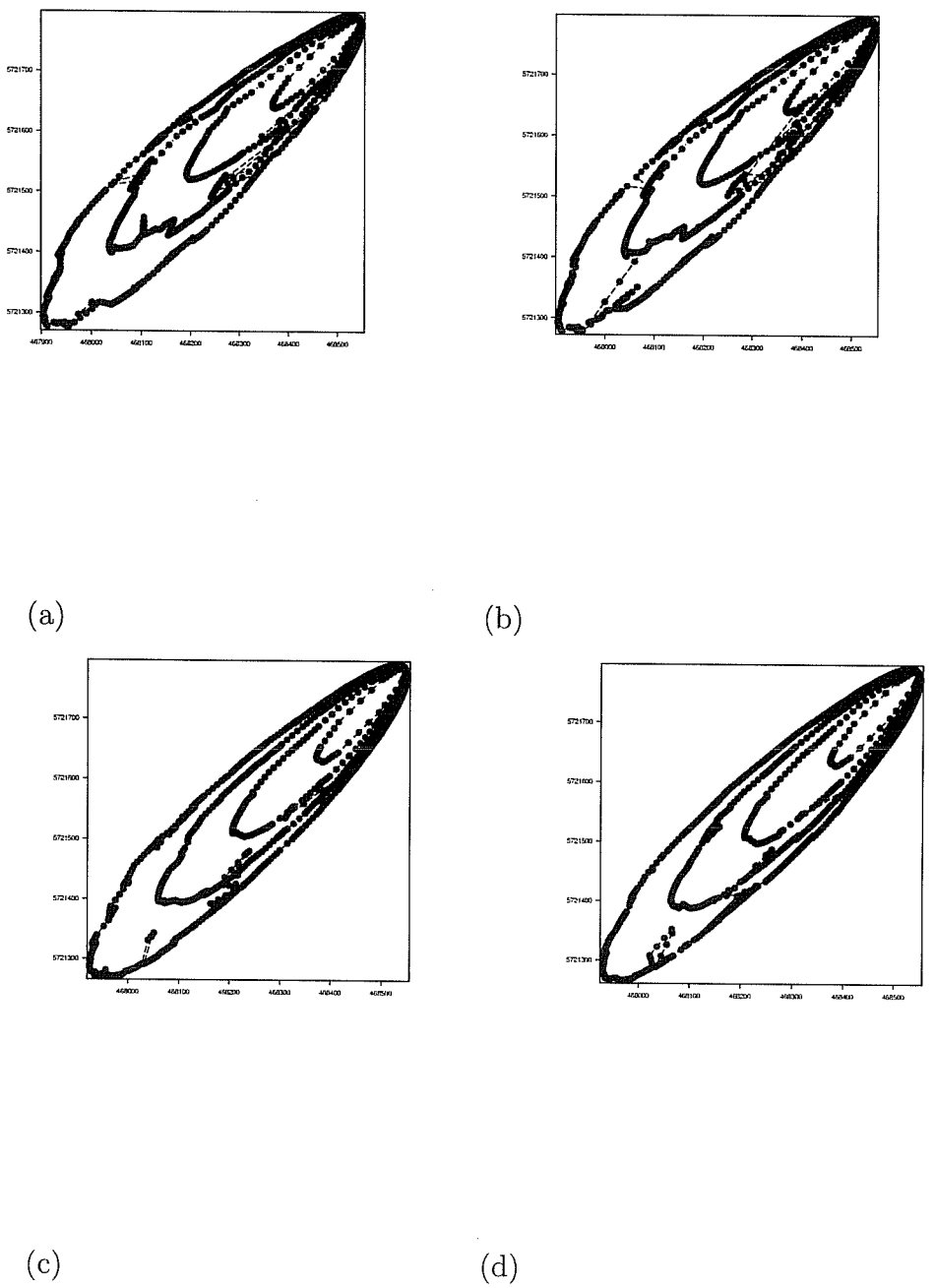
Based on the grid-resolution, l_g, divide the map into grid cells, where each grid cell is of size $l_g \cdot l_g$. Now, impose the following rules:

Rule 1 At most one vertex per grid cell. After a few initial steps (probably 2 or 3), we impose a rule of "at most one vertex per grid cell". That is, if several vertices are in the same grid cell, replace them by a vertex which is the average of vertices in the cell.

Rule 2 Minimum distance between vertices in adjacent cells $\geq 0.5 \cdot l_g$

The minimum distance between vertices in adjacent cells (cells sharing an edge) should be at least $0.5 \cdot l_g$, where l_g is the grid resolution. For each vertex, check its adjacent neighbors in the order of North, East, South and West. If the distance is less than $0.5 \cdot l_g$, replace the vertex by the average of the two vertices.

After Rule 2 is applied, if two vertices are still less than $0.5 \cdot l_g$ distance apart, they must be in two cells touching only at a corner, and that any other vertex must be at least at a distance $0.5 \cdot l_g$ from both of the vertices. Now apply Rule 3.



(a) (b)

(c) (d)

Figure 1.7: 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.

Rule 3 Minimum distance between vertices in diagonally opposite cells $\geq 0.5 \cdot l_g$.

If they are closer, replace them by the average. If the new vertex created lies in a neighboring cell with an already existing vector, replace both with the average of the two. Doing so preserves Rule 1. There are 4 diagonal neighbors to check: NE, SE, SW and NW.

Comments on Part 1

The above algorithm is implemented as follows: the cells are processed as one encounters them in the curve where one applies Rule 1 all the way through, then Rule 2 all the way through, and finally, Rule 3. In this manner, the rules will impose the constraints of their precedent rules.

Currently, it is uncertain the number of times Part 1 is implemented into the code. For thoroughness, it appears that one should implement the algorithm until no vertices have been shifted. This, however, can be computationally inefficient and counteract with the original purpose of this algorithm—improving efficiency. Thus, before implementing this algorithm, we must determine an iteration threshold.

Also, we are uncertain if the results of this algorithm will produce a fire front true to it's natural evolution. In replacing the vertices by averages, we may obtain a front that is entirely different from the front we originally believed to be true. To check the consistency of this algorithm, future work lies in implementing it and checking it's results.

Part 2: Automatic adjustment of time steps

Methodology:

If a vertex moved a long distance during a time step, divide that time step into smaller steps so that the vertex moves only by a "reasonable" amount during each of the smaller steps.

Assumption:

1. A "reasonable" length is a constant times the grid resolution, i.e. $c \cdot l_g$. For the current discussion, c is assumed to be 1. Denote this "reasonable" length by l_{maxs} (for length of maximum step). Note: a smaller c implies (hopefully) a better simulation, but at a cost of being slower.

2. We also assume that there is a routine such as the one discussed in Section 1.5.8 that can resolve any knot problems when given multiple fronts. This routine may reorder the vertices, move vertices, delete vertices, and insert vertices. With the constant c set to 1, we expect that the knot resolution routine to be somewhat simplified.

Phase 1: Approximate the new fronts

For each vertex, calculate its new positions, and at the same time calculate

a. the distance that each vertex has moved

b. the global maximum distance, d_{gmax}, travelled amongst all the vertices,

c. the global minimum distance, d_{gmin}, travelled amongst all the vertices.

If $d_{gmin} > l_{maxs}$, then the time step is too big for every vertex. Compute a new time step $t_{s_{new}} = \lfloor (t_s / \lceil (d_{gmin}/l_{maxs}) \rceil) \rfloor$. Repeatedly advance the time by $minimum(t_{s_{new}}, t_s - t_{s_{new}})$ until the current time is advanced by t_s.

Otherwise, (i.e. $d_{gmin} < l_{maxs}$), proceed to Phase 2.

Phase 2: Identify vertices of the fronts whose time steps should be reduced

For each front, start from initial vertex P_0^j, scan for the first vertex whose distance travelled is greater than l_{maxs}. If no such vertex exists for one front, the time step for this front does not have to be reduced. If all fronts are o.k., then pass all the fronts to the knot resolution routine.

Now, suppose there exists a vertex p^i whose velocity is too high. Select a set of consecutive vertices (we'll call this a **segment**) starting from vertex P_{i-1}^j, and ending at the first vertex P_k^j, $k > i$ whose distance travelled is at most l_{maxs}. If no such ending vertex exist, the whole set of vertices becomes the segment selected.

At this point, we will have a number of segments, each having the following variables:

$d_{max}(segment)$ = maximum distance travelled in time t_s amongst all vertices in the segment

$t_{inc}(segment)$ = time increment for this segment, and is equal to $\lfloor (t_s / \lceil (d_{max}(segment)/l_{maxs}) \rceil) \rfloor$

t_c = current time, which is initialized to 0

These segments also include the default segments whose vertices can all be advanced in one time step t_s without violating the l_{maxs} bound. The initial values for this segment are

$d_{max}(defaultsegments) \leq l_{maxs}$

$t_{inc}(defaultsegments) = t_s$

$t_c = 0$.

Phase 3: Advancing time

Next, we decide which segment is going to move first. This is the segment whose $t_c + t_{inc}$ is the smallest. All the vertices on this segment are advanced by a time equal to the $minimum(t_{inc}, t_s - t_c)$, and then t_c is incremented by this increment. The minimum calculation avoids overshooting the global time step t_s. While advancing the vertices in a segment, Rule 1 of Part 1 applies. If the new vertex lands in an occupied grid cell, the behavior depends on whether the occupying vertex's current time is equal to the new current time of the advancing segment.

case 1: occupying vertex is a new vertex of the current advancing segment. In this case, a weighted average is computed (the occupying vertex is weighted by the number of vertices that have been collapsed into the vertex while advancing the current segment).

case 2: occupying vertex is an older vertex (i.e., its current time is older). The advancing vertex stops at the boundary of the occupied grid cell.

Note 1: While advancing a segment, we treat the two end points of the segment as "processed", i.e., we assume its new position at the end of time step t_s is the one calculated from phase 1, and that its intermediate position is proportional to the elapsed time.

Note 2: Rules 2 and 3 of part 1 does not apply until the end of phase 3, when all segments are at time t_s.

Comments on Part 2

Before implementing this algorithm, we must consider its efficiency when compared with one that computes the worst global velocity and then stepping all vertices by this reduced time step

and appropriate number of steps. Currently, it is believed that while updating the vertices with a reduced and "safe" time step is simpler, the process becomes similar to the level set method.

Considering the efficiency of the algorithm proposed above, we make the following observation. Suppose that, with n vertices, 20% of the vertices should have their time step reduced by a factor of 10, that 60% reduced by a factor of 4, and 20% not reduced. If the time step of all vertices are reduced by a factor of 10, then we have $10 \cdot n$ units of work. If we use a variable time step, then we need $(1/5)n \cdot 10 + (3/5)n \cdot 4 + (1/5)n \cdot 1 = 4.6n$, which is a saving of a factor of 2. In its current form, we are uncertain whether this small saving factor is worth the complexity of the algorithm. We suggest looking at real data to verify its usefulness.

1.5.7 Regularization – II

Adaptive Vertices

While the implemented heuristic algorithm for adding vertices is somewhat successful, unnecessary vertices are not removed, leading to a stricter time stepping restriction, demonstrated by the crossing of vertices after a particular iteration. While these crossings occasionally fix themselves, one has to be wary of the solution, as the presence of an unphysical solution at any stage of the simulation usually makes results unreliable in general.

A simple algorithm to equi-distribute vertices after a few steps is likely a good, and cheap investment in the long run. Such an implementation will likely circumvent many of the vertex crossing problems. We begin by giving a quick background on monitor functions and equi-distribution, then we describe de Boor's algorithm as a possible numerical implementation. The benefits of de Boor's algorithm is its computational cost: $O(m)$, where m is the number of vertices on the front. First, a little background on equi-distribution.

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 "arc-length" monitor function,

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

where $y = u(x)$ is a function. In the example below, $\int_a^b \rho \, dx$ will result in the length of the curve. 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).

Equi-Distribution

Given some 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

$$\int_{x_k-1}^{x_k} \rho(u(x))\, dx = \frac{\int_{x_1=a}^{x_N=b} \rho(u(x))\, dx}{N} \quad \forall k = 1 \ldots N, \quad a \le x \le b. \tag{1.16}$$

de Boor's Algorithm in 1D

de Boor's algorithm is a simple, yet reliable algorithm that equi-distributes the vertices according to arclength, or some other monitor function. Suppose that the monitor function ρ is given on some prescribed set of vertices, $\{x_k\}$, $k = 0, \ldots, M$. Denote $\rho_k = \rho(x_k)$. The idea behind de Boor's algorithm is to approximate $\rho(x)$ by a piecewise constant functions based on the prescribed vertices,

$$\hat{\rho}(x) = \frac{1}{2}(\rho_{k-1} + \rho_k) \quad \text{for } x \in (x_{k-1}, x_k), \quad k = 1, \ldots, M, \tag{1.17}$$

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

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

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 (1.17). 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.

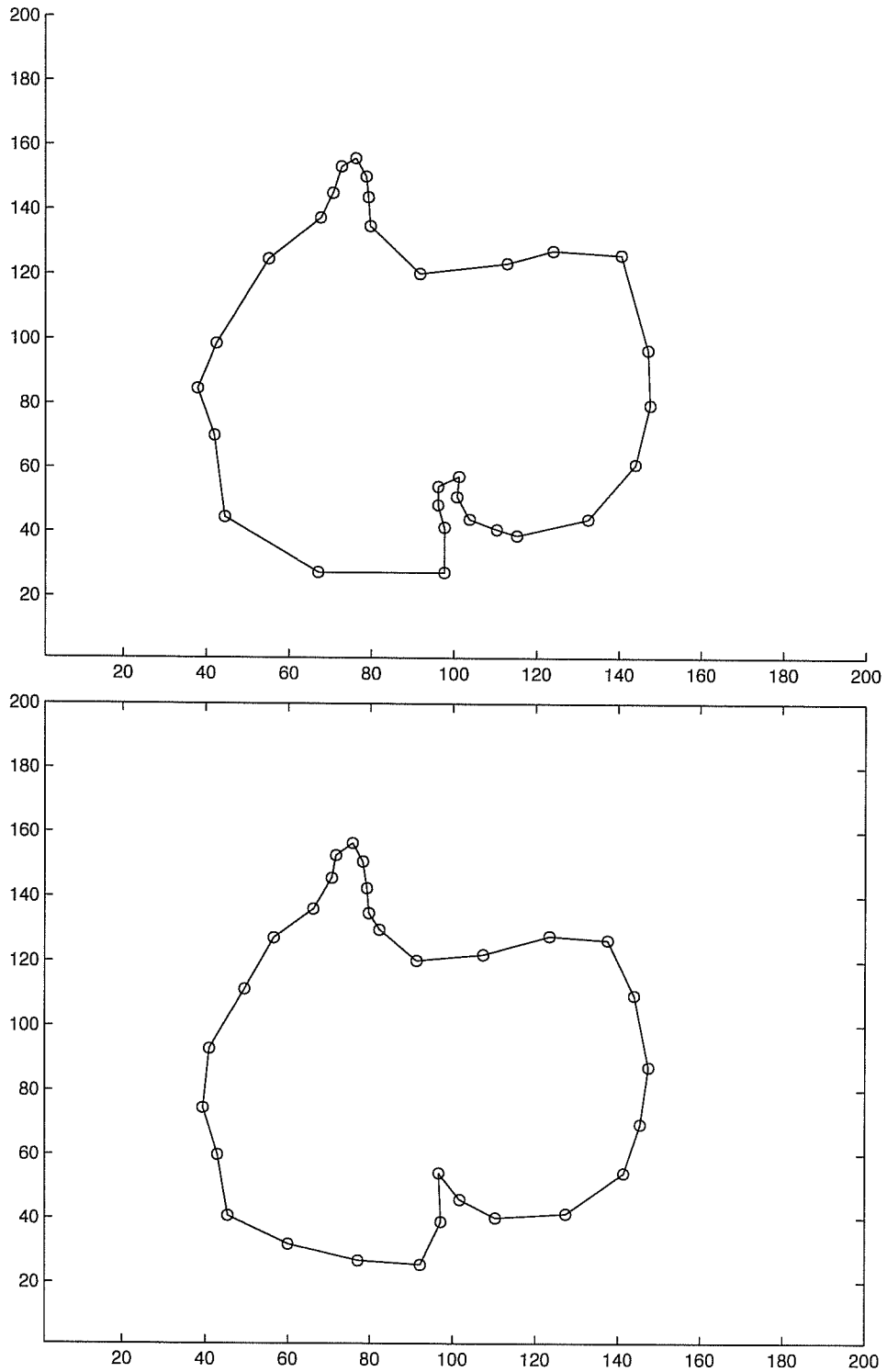


Figure 1.8: The top figure gives an example of propagated vertices, and the bottom figure shows equidistribution of vertices via a curvature based monitor function. Notice that there is some inherent smoothing due to the interpolation.

de Boors Algorithm for fronts

For equi-distributing on a 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

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

where s is some parametric parameter, and noticing that

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

where $d(\cdot, \cdot)$ is the distance between the two vertices, κ_j is the curvature at vertex x_j, v_j is some measure of the rate of spread at x_j, and $f(\kappa_j, v_j)$ is part of the monitor function used to evaluate the "importance" of the front segment. Now, to find $\{y_k\}$, $k = 1, \ldots, N-1$, 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 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 1.8

We note that modifications are required to handle vector barriers. The two natural approaches that come to mind are:

1. If the vertices on the front coincide with a barrier, fix those vertices with the vector edges of the barrier and equi-distribute around the rest of the front.

2. Equi-distribute around the entire front, then add the barrier edges as vertices.

1.5.8 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 1.9 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.

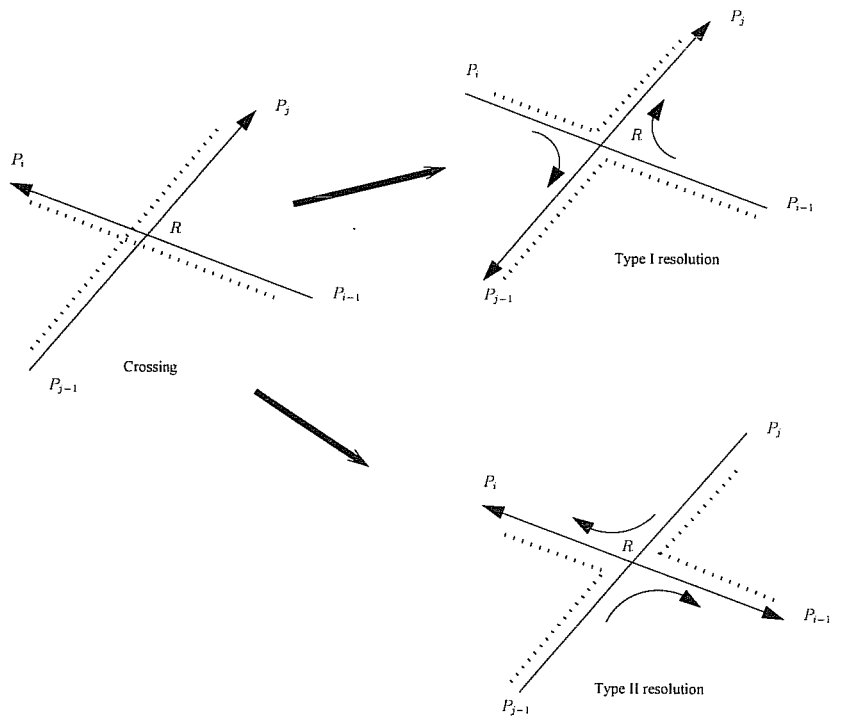


Figure 1.9: A crossing of the fire front with burned areas to the left.

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 $P_{i-1} \rightarrow P_i$ 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 1.11.

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

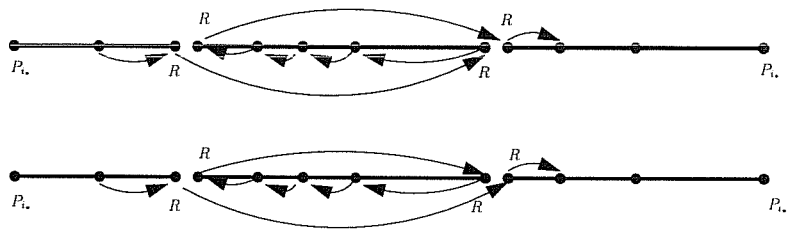


Figure 1.10: Resolution of crossing point – Data structure form. Crossing at vertex R

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 $P_{i_*-1} \rightarrow P_{i_*} \rightarrow P_{i_*+1}$ 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 $P_{j-1} \rightarrow P_j$ 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 1.10. Note that one resolution disconnects the curve into two closed paths (loops), but the other leaves a single path. In either case we have:

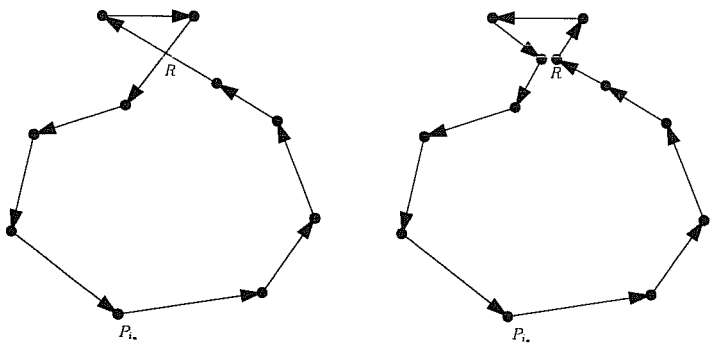


Figure 1.11: Merging an external loop

Proposition 1 *In the case of crossing resolutions of Type II, the new path segment* $P_{j-1} \rightarrow R \rightarrow P_{i-1}$ *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 1.10.

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 $R \rightarrow P_{j-1}$. 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.

1.5.9 Examples

- **Merging an external loop** In Figure 1.11 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 Section 1.5.2 and Figure 1.5.

- **Clipping an internal loop** In Figure 1.12 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

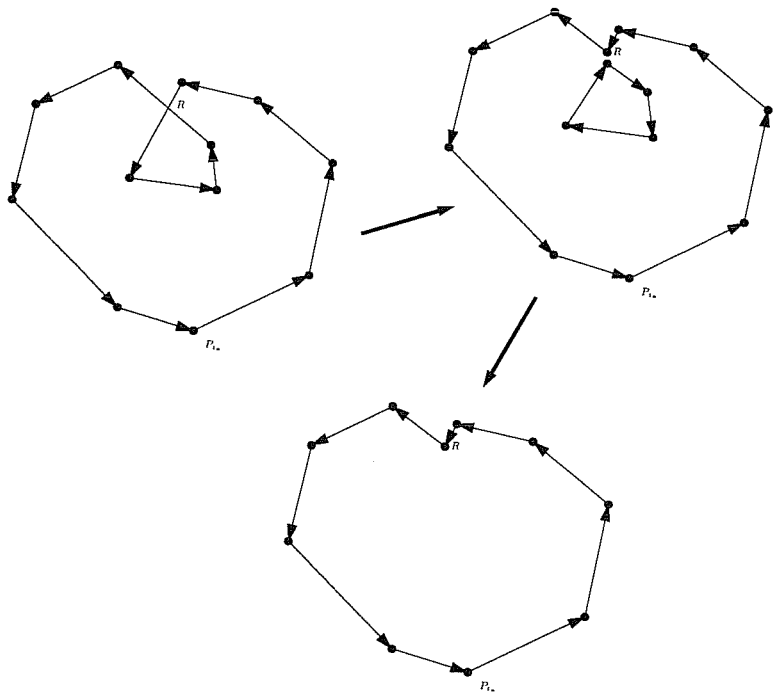


Figure 1.12: Clipping an internal loop

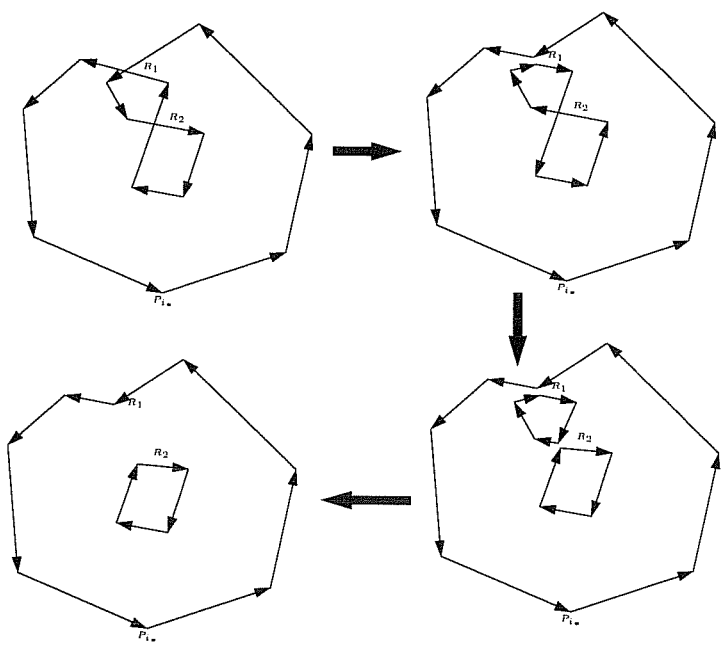


Figure 1.13: Horseshoe fire

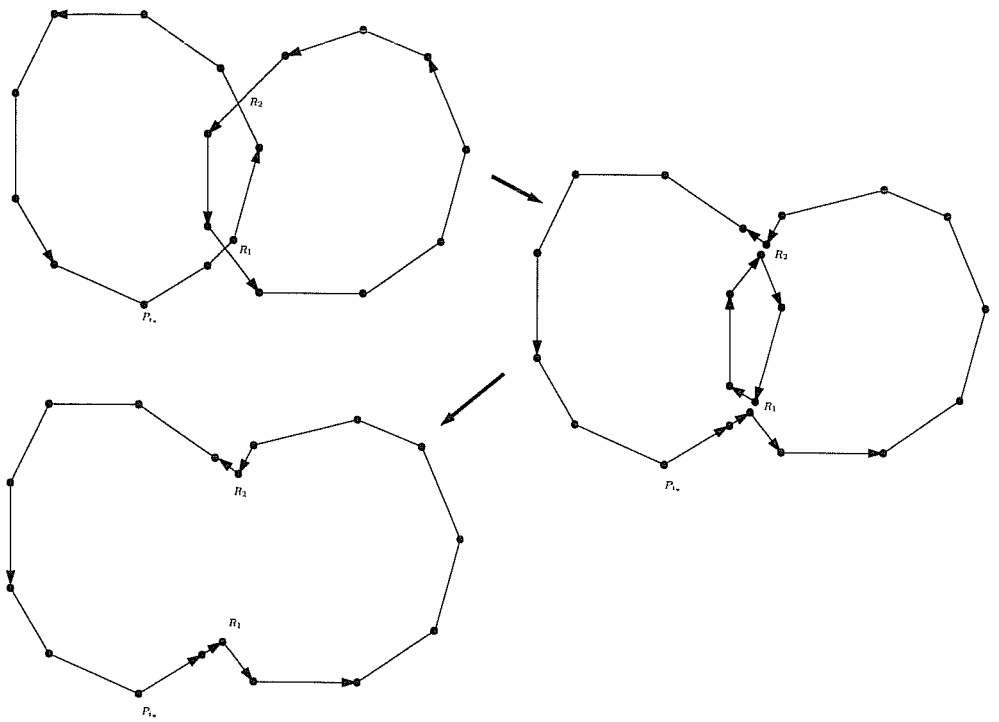


Figure 1.14: Merging of two simple fire fronts

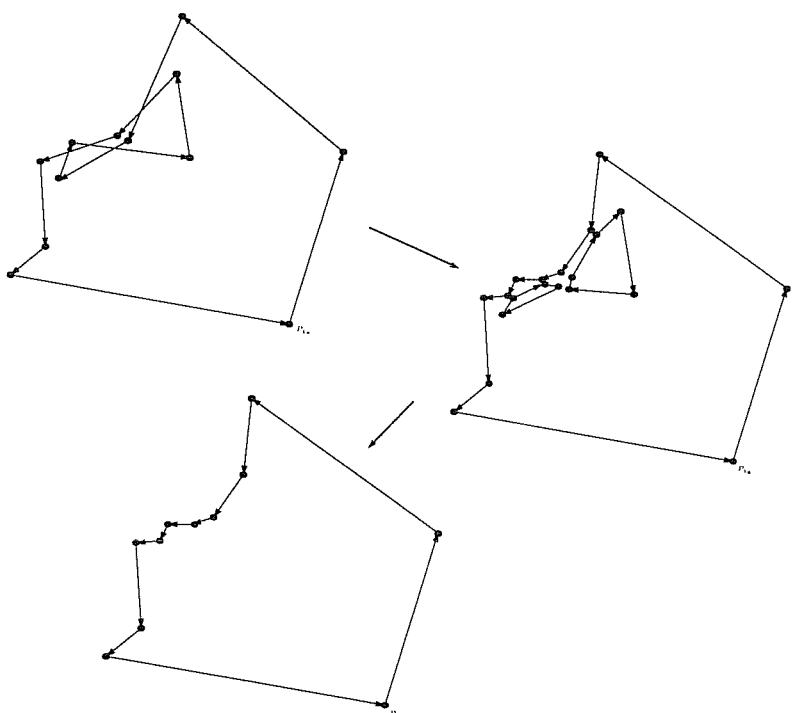


Figure 1.15: Resolution of the PROMETHEUS nightmare example

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.

- **Horseshoe fire** In Figure 1.13 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.

- **Merging fires** In Figure 1.14 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.

- **Prometheus nightmare** This example was presented by PROMETHEUS developers in [12] 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 1.15 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.

1.5.10 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.

1.6 PROPAGATION BY LEVEL SETS

The *Level-Set* method is a state-of-the art numerical algorithm to handle interface propagation for a variety of applications such as the simulation of multiphase flows and image processing. Its underlying principle is very simple and is easily adapted for new problems. Our goal here is to demonstrate that the Level-Set method is a very efficient tool to simulate 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 approach: merging of distinct fire fronts into one front, selecting a time-step automatically, including fire propagation barriers (for example, rivers). The goal here is to test the level-set approach on benchmark test-cases from the PROMETHEUS database. For that purpose, we will describe how to interpret the elliptical fire propagation model data in the context of the Level-Set method. We will then assess the performance of the approach by comparing the results with those of the PROMETHEUS benchmark tests.

1.6.1 Basic principle for the *Level-Set* method

The basic principle of the *Level-Set* method is very simple. It leads to a formulation in terms of a Hamilton-Jacobi partial differential equation (PDE), that is typically solved numerically. This PDE is nonlinear and can develop singularities. This means great care must be taken to numerically discretize the solution, as stability and accuracy can only be achieved using some form of *upwinding*. This is a well-studied issue; we will not go into the details in this report, but instead refer to book [5] where the topic is covered exhaustively.

Let's 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 of 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 speed of propagation of the local iso-level of $\phi(x, y, t)$ so that it coincides with the fire front speed on the zero-level. See [5] to extend the zero-level velocity to the rest of the domain if this later one is not readily available.

The evolution equation for ϕ is obtained as follows. The curve $\Gamma(t)$ that represents the front at time t is given by

$$\phi(x(t), y(t), t) = \phi(\vec{r}(t), t) = 0. \tag{1.18}$$

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

$$\phi_t + \nabla\phi \cdot \frac{d\vec{r}(t)}{dt} = 0. \tag{1.19}$$

which is equivalent to

$$\phi_t + \vec{v} \cdot \nabla\phi \cdot = 0. \tag{1.20}$$

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 $\Gamma(0)$. 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. $\phi(x, y, 0) = \sigma(x, y)d((x, y), \Gamma(0))$, where d is the shortest distance between (x, y) and $\Gamma(0)$ and

$$\sigma(P) = \begin{cases} -1, & \text{if } (x, y) \text{ is inside } \Gamma(0) \\ 0, & \text{if } (x, y) \text{ is on } \Gamma(0) \\ 1, & \text{if } (x, y) \text{ is outside } \Gamma(0). \end{cases} \tag{1.21}$$

1.6.2 Handling the challenges stemming from the ellipse approach

Front merging

The topological changes that occur when two fronts merge can cause a lot of logistic problems in managing the data from the ellipse approach. On the other hand, the Level-set method does not require any particular treatment to handle such a case. 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, so that this merging causes no particular problem for the Level-Set method.

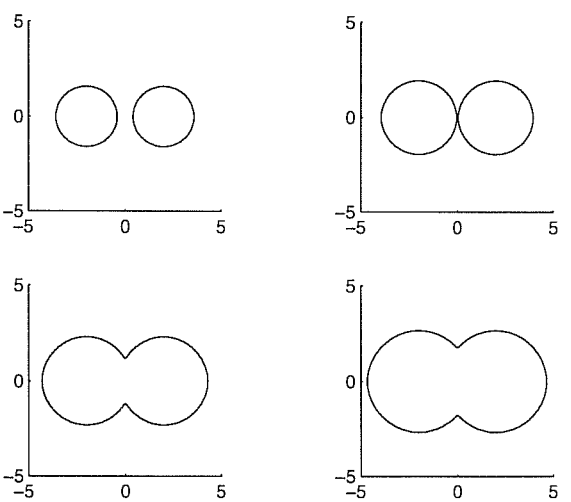


Figure 1.16: Merging of two circles

Selecting an appropriate time-step

The Level-set method in this case requires solving a the unsteady PDE 1.20 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 theory of the numerical integration for Hamilton-Jacobi equations requires that the time-step Δt satisfies a stability condition (related to the "CFL" condition) of the following form:

$$\max_{domain} \{v_1, v_2\} \Delta t \leq \min\{\Delta x, \Delta y\} \tag{1.22}$$

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

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 it. To take into account the presence of such obstacles in the level-set method, it is sufficient to impose that the propagation speed is strictly zero for all mesh nodes directly surrounding the barrier. Figure 1.6.2 shows those points for one example. Figure 1.6.2 illustrates the good performance of the level-set solution as it propagates around the obstacle, with the two ends of the front eventually merging on the other side of the obstacle.

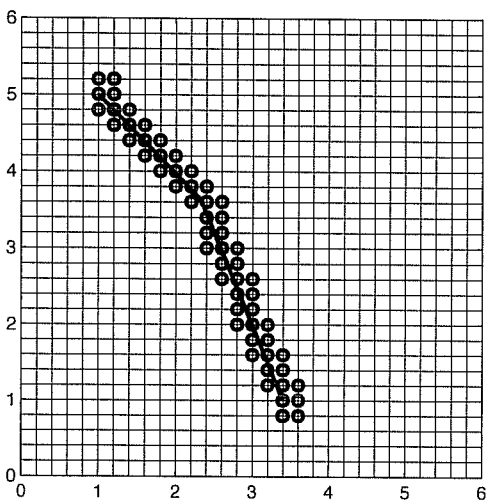


Figure 1.17: Nodes surrounding a barrier where the propagation speed is set to zero.

1.6.3 Level-set speed of propagation based on the PROMETHEUS database

Here is the procedure to convert the ellipse burning rates from the Prometheus database into a propagation speed to be used with the level-set method.

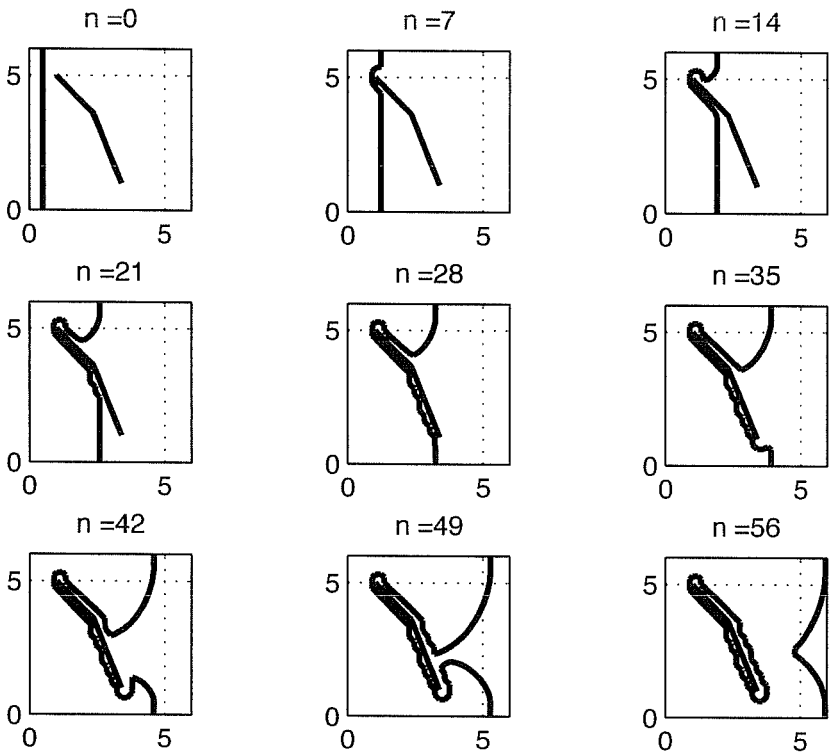


Figure 1.18: Example of propagation with a barrier.

First we recall the (continuous) front evolution equation (1.8) from the end of Section 1.3 defines $\vec{v}$:

$$\vec{v} = \begin{pmatrix} x_t \\ y_t \end{pmatrix} = \frac{A^T A\vec{n}}{\|A\vec{n}\|} + \vec{c} \tag{1.23}$$

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 (1.2). The vector $\vec{n} = (y_s, -x_s)$ was previously used for the correctly oriented unit normal to the curve but we can just as well take $\vec{n} = \frac{\nabla\phi}{\|\nabla\phi\|}$. Substituting this and equation (1.23) into equation (1.20) gives the following

$$0 = \phi_t + \|A\nabla\phi\| + \nabla\phi \cdot \vec{c} \tag{1.24}$$

The term $\nabla\phi \cdot \vec{c}$ is the advection term and

$$\|A\nabla\phi\| = \|A\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\| \, \|\nabla\phi\|$$

is the self-propagation term in the normal direction. Setting

$$F(x,y,t) = \|A\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|$$

the norm of the normal speed of the front,

$$\phi_t + F(x, y, t)\|\nabla\phi\| + \nabla\phi \cdot \vec{c} = 0. \tag{1.25}$$

This is a very good (order dt) approximation of the propagation speed.

1.6.4 Simulation results

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 rectangle where the speed of propagation is higher. It should be noted that, due to time constraints during the workshop, the simulations were run with velocity data $\vec{v}$ at equation (1.23) provided by the PROMETHEUS engine, rather than a full implementation of the level set equations (1.25) on the parametric data.

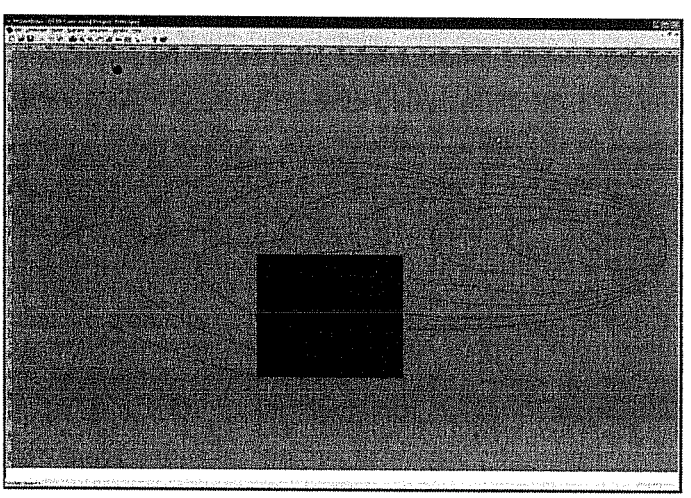


Figure 1.19: Simulation via PROMETHEUS (screenshot)

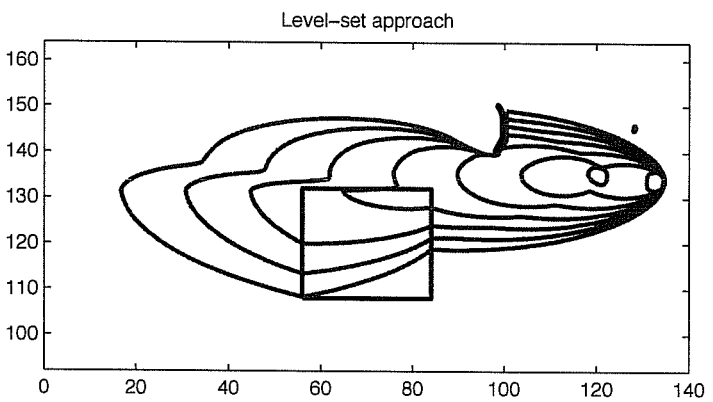


Figure 1.20: Simulation with the level-set method

There is remarkable 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. There are some minute differences, but overall, there is excellent agreement.

1.7 Conclusions and Suggestions for Future Work

In Sethian's foundational book [5] 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.

The approaches considered in this document include all three of these known methods. We briefly summarize.

1.7.1 Smoothing

The approach used here was to smooth the parametric data defining the velocity field in equation (1.6) 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.

1.7.2 Redistribution and Adapted Timesteps

While the PROMETHEUS code already allows for redistribution of vertices, any automatic or even more systematic way of doing this would be of great interest to the developers. It should be noted that the goal in redistribution is to keep the timestep from shrinking to zero when the numerical method attempts to avoid crossings and tangles.

1.7.3 Delooping and Clipping

The outer-hull approach investigated during the workshop now appears, in retrospect, to be similar to techniques used by other fire modelling software. This should be investigated thoroughly. PROMETHEUS has coded up and tested the algorithm suggested in Section 1.5.8 and reports 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 continued 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 crossing resolutions seems mathematically feasible and a good first step to understanding this new approach.

1.7.4 Level Set Approach

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 first 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, including in very heterogeneous landscapes and very unsteady conditions as well as incorporating stochastic effects.

1.8 APPENDIX – More about Level Sets[11]

1.8.1 Level Set Framework

Whenever one has topological changes - in this case, two fires merging to make a larger fire, a natural approach is to use a level set framework, introduced by Osher et al. [8]. Additionally, if there are instabilities in the numerical method (such as the simple or figure eight crossings observed in the PROMETHEUS code), level set methods avoid the problems altogether by representing the front as an implicit surface. Our discussion here is slightly different from the work that Anne Bourlioux, Eric Brunelle and Chris Bose derived. Namely, they were interested in using given ROS, BROS and FROS data to derive a front velocity. Our approach is more simplistic, but was not pursued due to lack of time and support for the idea.

The resistance between changing from a Lagrangian approach (the marker particle framework currently employed in PROMETHEUS) and an Eulerian approach (the proposed level set framework) is understandable, and expected as it involves a re-design of the entire structure of the PROMETHEUS code. We address some of the assumed/perceived difficulties before describing the framework, equations and algorithms.

Perceived/assumed problems for people unfamiliar with the inner workings of the level set framework

- *Isn't using the level set framework more expensive?* Yes, it is more expensive, but it is still $O(m)$, where m is the number of grid nodes close to the front, provided a banded level set approach or a fast marching method is employed.

- *How much memory will this take?* Significantly more, since we will have to store information on the entire domain, even if we do not involve them on the computation. A simulation with 20,000+ data points on an every day laptop is not unreasonable now adays, however. If memory constraints are an issue, one can choose to store only information close to the front, and "extend" that information as necessary. This, of course, increases the computational complexity.

- *But you can't represent vector breaks!* Not entirely true. If necessary, the level set framework can be laid on a non- uniform grid, say a finite element mesh. It is computationally more expensive as one loses the benefits of a uniform mesh. Another approach is to use an "adaptive level set" function that clusters more grid points around the vector 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.

- *Can we retrieve vector based data?* Absolutely. The beauty of level set functions is their ability to flag regions in the domain as burned, on the front, or an unburned region. A simple interpolation of the level set function will give vector data at any instance in the simulation that can be used in GIS calculations.

[11] Appendix contributed by Ben Ong

Implicit Surfaces

The foundational idea that drives level set based algorithms is the implicit representation of lower dimensional interfaces. Consider a region of space $\Omega \in R^2$ enclosed by a front. We embed this front $\partial\Omega$ in a *level set function*, $\psi(\vec{x})$.

$$\begin{cases} \psi(\vec{x}) = 0 & \vec{x} \text{ on } \partial\Omega \\ \psi(\vec{x}) < 0 & \vec{x} \in \Omega \quad \text{(burned region)} \\ \psi(\vec{x}) > 0 & \vec{x} \notin \Omega \quad \text{(unburned region)} \end{cases} \tag{1.26}$$

This idea is better shown through the next example:

$$\psi(\vec{x}) = \psi(x, y) = x^2 + y^2 - 1$$

$$\partial\Omega = \{\vec{x} \,|\, \psi(\vec{x}) = 0\} = \{x^2 + y^2 = 1\}$$
$$\Omega = \{\vec{x} \,|\, \psi(\vec{x}) < 0\} = \{x^2 + y^2 < 1\}$$

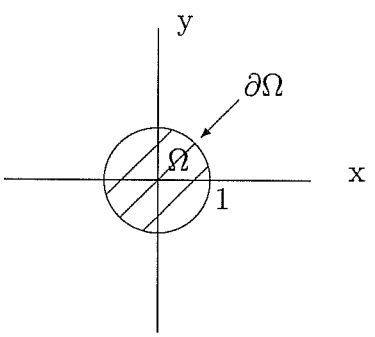


Figure 1.21: Ω is the region inside the circle. N.B., we call this a closed region because there is a clear interior and exterior region.

Evolution Equation

To derive an evolution equation for the level set function, we require that a particle on the zeroth level set of ψ remain on the zeroth level set at a later time. Mathematically,

$$\psi\left(\vec{x}(t), t\right) = 0.$$

By the chain rule,

$$\psi_t + \nabla\psi\left(\vec{x}(t), t\right) \cdot \vec{x}'(t) = 0,$$

or equivalently,

$$\psi_t + \vec{V} \cdot \nabla\psi = 0, \tag{1.27}$$

Equation (1.27) is often referred to as the *level set equation.* Notice that the level set equation falls into the general class of Hamilton-Jacobi equations.

Front Velocities

Consider a completely flat surface, homogeneous material with no wind. A wildfire front will move with constant normal velocity, where the rate is dependent on the fuel type. Lets call this velocity on the front $\vec{v}_1(x, y)$. Note that for the entire domain, we have a "normal" direction coming from level contours of our level set function, namely $\nabla\psi$, so we can define our velocity $\vec{v}_1(x, y)$ on the entire domain. Non homogeneous material is easily modelled

It is unclear to me how wind will affect the front velocities. Two ideas seem reasonable. The first model is to treat the wind as a simple advective term, $\vec{v}_2(x, y)$ which is additive with $\vec{v}_1(x, y)$ as defined above. The second model is to also consider the non-linear effects from having the wind "closer" to the ground due to the wind. Presumably, a wind blowing in the direction of the normal will speed up the front propagation due to the pre-heating, and wind blowing opposite to the normal will slow front propagation.

Now, if we account for terrain variation, one should be able to derive a third component to the velocity, $\vec{v}_3(x, y)$, which takes the normal direction from level contours and the terrain data into account. This will likely result in something similar to the WSE (Wind Speed Equivalent) model currently employed Prometheus. We may also have to project velocities $\vec{v}_1(x, y)$ and $\vec{v}_2(x, y)$ to account for terrain variation.

We then combine all three components to form the composite velocity of the front. Note that the normal speed always has to be non negative to ensure that the flame does not move back into a "burned" region.

Initialization of the level set function

Given a boundary $\partial\Omega$, how do we define the level set function ψ? A desirable property of level set functions is smoothness; we thus impose (in addition to the implicit representation) the condition $|\psi(\vec{x})| = d(\vec{x})$ where $d(x)$ is the shortest distance to the boundary, i.e.,

$$d(\vec{x}) = min(|\vec{x} - \vec{x}_I|) \quad \forall \vec{x}_I \in \partial\Omega.$$

The resulting level set function is then known as a signed distance function. It also holds (in a general sense) that

$$|\nabla\psi(\vec{x})| = 1.$$

A naive way to initialize a signed distance function is to evolve equation (1.28) until a steady state solution for ψ is reached.

$$\psi_t + |\nabla\psi| = 1. \tag{1.28}$$

Upon examination however, equation (1.28) propagates information from lower values of ψ to larger values of ψ. This means that the boundary evolution will be influenced only by the negative values of ψ. A better initialization equation that circumvents this problem is

$$\psi_t + sgn(\psi_0)\,(|\nabla\psi| - 1) = 0, \tag{1.29}$$

where $sgn(\psi_0)$ is the sign function which takes the values

$$sgn\left(\psi_0(\vec{x})\right) = \begin{cases} 1 & x \in \Omega \\ -1 & x \notin \Omega \\ 0 & x \in \partial\Omega. \end{cases} \tag{1.30}$$

Equation (1.29) is also known as the reinitialization equation; when evolving the level set functions, one often cares exclusively about the zero contour which separates the interior and the exterior of the domain. In the course of a computation, the level set function may stray from the smoothed signed distance formulation. In that case, the reinitialization equation (1.29) is used to bring the level set back to a signed distance function.

In discretizing (1.30), studies have shown that better results are obtained when a smoothed out sign function $sgn(\psi_0)$ is used.

$$sgn(\psi) = \frac{\psi}{\sqrt{\psi^2 + |\nabla\psi|^2(\Delta x)^2}},$$

Discretization of the level set equation

Since level set equations are a special class of Hamilton-Jacobi equations, We discretize the level set equation spatially using an upwinding scheme, and solve the resulting time ODEs using the Dormand Prince Runge-Kutta 4-5 Scheme (7 steps) with adapting time stepping. If higher spatial accuracy is desired, a higher order scheme, like an ENO (Essentially Non Oscillatory scheme) or a WENO scheme (Weighted Essentially Non Oscillatory) can be employed.

Spatial Discretization

Recall the level set equation 1.27

$$\psi_t + \vec{V} \cdot \nabla\psi = 0. \tag{1.31}$$

In one dimension,

$$(\psi_t)_i + v_i(\psi_x)_i = 0. \tag{1.32}$$

where i is the index for spatial discretization.

If $v_i > 0$, the values of ψ are moving from left to right, and the method of characteristics tells us to look to the left of x_i to determine ψ_x. Similarly, if $v_i < 0$, the values of ψ are moving from right to left, and the method of characteristics tells us to look to the right of x_i to determine ψ_x. This is of course just upwinding (a first order ENO scheme).

$$(\psi_x^+)_i = \frac{\psi_{i+1} - \psi_i}{x_{i+1} - x_i}, \quad (\psi_x^-)_i = \frac{\psi_i - \psi_{i-1}}{x_i - x_{i-1}}.$$

$$\psi_x = \begin{cases} (\psi_x^+)_i & \text{if } v_i < 0, \\ (\psi_x^-)_i & \text{otherwise} \end{cases}$$

This first-order accurate upwind scheme can be improved upon by using a more accurate approximation for ψ_x^+ and ψ_x^-. In higher dimensions, ψ_x, ψ_y can be calculated in a *dimension-by-dimension* manner.

Runge Kutta methods

To solve the resulting ODEs after a spatial discretization, a Dormand Prince 4-5 (7 step) algorithm with variable time stepping can be employed. The idea is to take intermediate steps to help cancel out lower order terms.

$$\begin{aligned}
\vec{k}_1 &= f(t, \vec{\psi}) \\
\vec{k}_2 &= f(t + c_2\, dt, \vec{\psi}(t) + dt(a_{2,1}\vec{k}_1)) \\
\vec{k}_3 &= f(t + c_3\, dt, \vec{\psi}(t) + dt(a_{3,1}\vec{k}_1 + a_{3,2}\vec{k}_2)) \\
&\dots \\
\vec{k}_7 &= f(t + c_7\, dt, \vec{\psi}(t) + dt(a_{7,1}\vec{k}_1 + a_{7,2}\vec{k}_2 + \dots + a_{7,6}\vec{k}_6)) \\
\vec{\psi}(t + dt) &= \psi(t) + dt(b_1\vec{k}_1 + \dots b_s\vec{k}_s)
\end{aligned}$$

where

$$\begin{aligned}
\vec{b} &= \left[\frac{35}{384}, 0, \frac{500}{1113}, \frac{125}{192}, -\frac{2187}{6784}, \frac{11}{84}, 0\right], \\
\vec{c} &= \left[0, \frac{1}{5}, \frac{3}{10}, \frac{4}{5}, \frac{8}{9}, 1, 1\right],
\end{aligned}$$

$$\begin{aligned}
a_{2,1} &= \frac{1}{5}, \\
a_{3,1} &= \frac{3}{40}, & a_{3,2} &= \frac{9}{40}, \\
a_{4,1} &= \frac{44}{45}, & a_{4,2} &= -\frac{56}{15}, & a_{4,3} &= \frac{32}{9}, \\
a_{5,1} &= \frac{19372}{6561}, & a_{5,2} &= -\frac{25360}{2187}, & a_{5,3} &= \frac{64448}{6561}, & a_{5,4} &= -\frac{212}{729}, \\
a_{6,1} &= \frac{9017}{3168}, & a_{6,2} &= -\frac{355}{33}, & a_{6,3} &= \frac{46732}{5247}, & a_{6,4} &= \frac{49}{176}, & a_{6,5} &= -\frac{5103}{18656}, \\
a_{7,1} &= \frac{35}{384}, & a_{7,2} &= 0, & a_{7,3} &= \frac{500}{1113}, & a_{7,4} &= \frac{125}{192}, & a_{7,5} &= -\frac{2187}{6784}, & a_{7,6} &= \frac{11}{84}.
\end{aligned}$$

Bibliography

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

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

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

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

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

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

[7] 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).

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

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

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

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

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

[13] 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`

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