MATH 238
DIFFERENTIAL EQUATIONS
LAURA MOORE-MUELLER
GREEN RIVER COLLEGE

PATHFINDING IN TWO DIMENSIONS USING ORDINARY DIFFERENTIAL EQUATIONS

May 20, 2015

Timothy Moore
Jason Murray
Carlie Olson
Cedrick Cooke

Abstract

This paper focuses on solving pathfinding problems in two dimensions without requiring grids or depending on graph theory. A method for creating continuous hazard functions that describe a problem space is derived, and pathfinding solutions are generated by following the slope of such hazard functions after beginning at an initial position. Pathfinding problems with an arbitrary number of barriers are converted into a system of first-order non-linear differential equations, which can be solved numerically to achieve a parametrized curve that describes the solution path.

Contents

Figure 1: Green: Starting position, Red: Ending position (Xu, 2014)

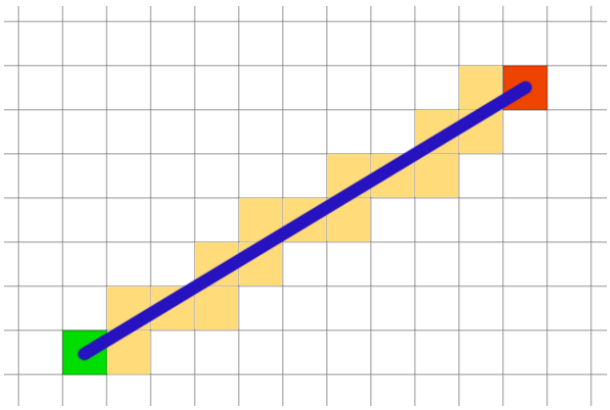
1 INTRODUCTION

Consider Figure 1. How can the best path from the green node to the red node be determined, provided that movement is only allowed in the cardinal directions? Humans can look at this problem and easily declare that the best path is the one generated by drawing a line between the two nodes, and filling in the squares that the line passes through. This is what has been done in Figure 2.

It is very difficult, however, to distill intuition generated by human minds about best-path problems. Differing procedures are used depending on the given problem, so creating an algorithm that is able to efficiently solve a wide variety of best-path problems is difficult, and is a topic of current computer science and software development research.

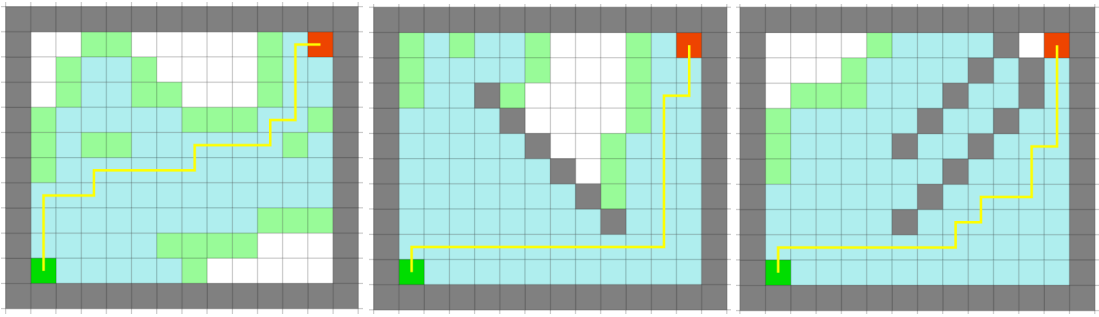
The traditional algorithm for time-efficient pathfinding in games is called A* (pronounced "A-Star") (Lucas et al., 2013, p. 4). It finds an optimal solution path by traversing nodes and

Figure 2: A Human-Solved Pathfinding Problem



assigning each a cost. Cost is generated by applying a heuristic[1] to the node, then adding the minimum number of nodes needed to travel to the node being considered. Choosing the right heuristic can greatly increase the efficiency of the algorithm—sensible node costs allow A* to examine as few nodes as possible. Any obstacles are marked as non-traversable and are essentially given indefinitely high cost. See Figure 3 for some examples of A* in action.

Figure 3: Example A* Solutions to Varying Problem Grids (Manhattan Distance Heuristic) where Gray: Walls, Green: Nodes Considered, Blue: Paths Explored, (Xu, 2014)



Notice how adding obstacles increases the amount of paths that are explored and expanded on; memory and computation time become a bottleneck for sufficiently complex problem grids, especially when the results are needed for real-time applications (Lucas et al., 2013, p. 21). Memory efficiency is poor because the costs for all of the nodes that have previously been inspected must be stored in memory while the algorithm is executing, and computation time-efficiency is poor because time must be spent to explore multiple paths, most of which aren't included in the final solution path. These inefficiencies become more apparent when problem grids vary with time, because the pathfinding algorithm must be executed again **every time the scene changes**.

Various examples of real-time applications for pathfinding algorithms include global positioning systems (GPS) and video game artificial intelligence (AI) (Lucas et al., 2013, p. 21; Algfoor, Sunar, and Kolivand, 2015, p. 1). Efficient and correct pathfinding algorithms reduce the amount of time a GPS unit spends "recalculating" when the user goes off-route, and enemies in video games typically employ some kind of pathfinding algorithm in order to approach the player in a realistic and advantageous way. Consider, though, that when a scene changes, a new path must be calculated. Not only must the algorithm be re-applied, but the grid representing the problem must also be regenerated. This is a problem that is still holding back AI in video games today (Graham, McCabe, & Sheridan, 2003, p. 13). Therefore, the

[1]Heuristics are methods used to determine a reasonable cost for a node. For example, the Manhattan Distance heuristic generates cost based on the sum of the horizontal and vertical distances from the node examined to the destination node.

efficiency of various algorithms plays a large part in decisions regarding pathfinding. While the efficiency of the method derived in this paper will not be thoroughly examined herein, further research into its efficiency could provide useful insights.

1.1 Differential Equations in Pathfinding

Differential equations present a novel but potentially useful way of viewing pathfinding problems, and their use in pathfinding appears to be relatively unexplored in current academic research. Engebretsen (2014, p. 25) very recently used a fluid flow analogy in his master's thesis, reducing all of the physical functions involved in a fluid flow equation into one "hazard" function H. A hazard function varies over x and y, and solving a fluid flow partial differential equation provides a potential field from which streamlines can be extracted; those streamlines are candidate solution paths.

Another mathematical pathfinding method tailored for Unmanned Aerial Vehicles was proposed by Babel (2012, p. 257). In his method, the solution is described as a parametrized curve. In our problem derivation below, we combine the ideas of a hazard function and a parametrized solution to come up with an initial value problem whose solution solves pathfinding problems. However, instead of using streamlines, we will be using pathlines derived from a hazard function.

2 DERIVATION

2.1 The Hazard Function

The hazard function H describes the problem grid. This model generates a solution based on the path that a metaphorical ball would take if it were to be placed at an arbitrary point on H. Thus, the model requires the generation of a hazard function that yields a useful path. The actual solution to the pathfinding problem is calculated by using differential equations, which can be solved numerically. Like Engebretsen (2014, p. 25), the tendency for the path to go over a location will be referred to as hazard.

For this initial derivation, assume that hazard depends on location and does not vary with time. Consider again Figure 1. Let the value H at an arbitrary point (x, y) represent how compelled a metaphorical ball is to move towards that point. Clearly, the most compelling place to be is the destination, which implies that other places are less compelling. Since, for visualization purposes, the solution path will match the path that a ball would take if it were

to be dropped into the map from an initial position, undesirable positions will be defined to yield large values for H and desirable positions will yield lower values for H.

Differential equations require continuous functions, so we will derive H such that it is at least twice continuous over some interval I which contains the starting point, the ending point, and a sufficient radial buffer zone for the path. This buffer zone is required because the final path will, in most cases, not be a straight line; if there are large obstacles that must be avoided, the path could veer arbitrarily in any direction.

The solution will follow the hazard graph as a ball rolling down a hill, so it is important that the ball finishes rolling at the desired end location. It is sufficient to say a ball will go to the lowest hazard if there are no other flat locations along the hazard map, and the hazard map tends towards the lowest location.

This analogy assumes that the reason the ball avoids obstacles and goes toward the end location is to avoid hazard, so it is sensible to assume that hazard increases as the distance from the goal location increases. For a convenient parabolic shape, suppose hazard increases as distance squared from the goal location increases. Let (x, y) be the position of the ball at time t, and (x_{goal}, y_{goal}) be the constant goal location of the ball.

$$H(x,y) = (x - x_{goal})^2 + (y - y_{goal})^2 \tag{1}$$

Using equation 1, if an object is dropped at any point and allowed to slide, it will *always* end up at the point of lowest hazard, which will *always* be at (x_{goal}, y_{goal}). Furthermore, the path will follow a straight line towards the goal since no boundaries are included on the hazard map. In order to implement boundaries, the hazard near the boundary must be large, and the hazard away from the boundary must be low. However, the hazard caused by the boundary cannot be negative or be too close to the goal, otherwise it may significantly alter the final resting location of the ball.

One convenient function with these properties is the reciprocal of the distance squared from the hazard point. This will quickly approach 0 but will not be negative as distance increases, and will increase rapidly near the hazard point.

$$H(x,y) = (x - x_{goal})^2 + (y - y_{goal})^2 + \sum_{n=0}^{N_{hazards}} \frac{1}{(x - x_{hazard_n})^2 + (y - y_{hazard_n})^2} \tag{2}$$

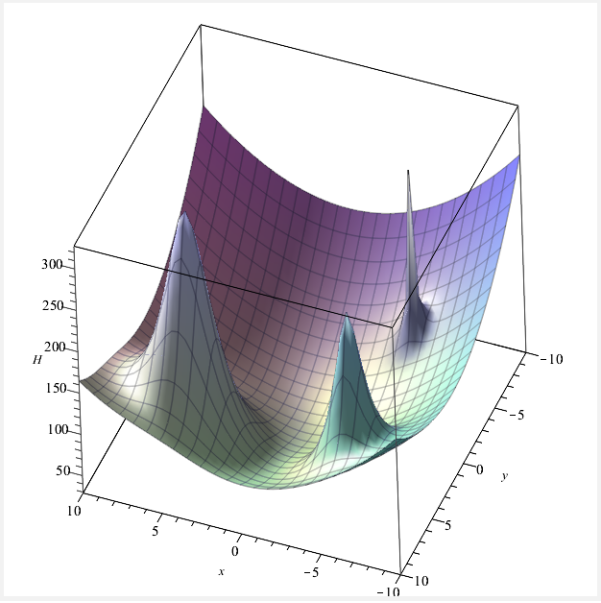
However, this function is not continuous over the full region I, particularly where the distance to the hazard point is 0. Since the distance squared is never negative, the denominator becomes strictly positive by including a small minimum in the denominator. A fraction with

a constant numerator is greatest when the denominator is smallest. The denominator is smallest when the distance squared to the point is 0. Thus, the largest value of the hazard is the reciprocal of the new term, and thus it is sensible to refer to this term as how afraid the ball is of touching that point, or fear (f) for short. It is more intuitive to have the hazard at the boundary go up as fear goes up, so the reciprocal of the fear of the hazard is used to undo the reciprocal required to get the maximum.

$$H(x,y) = (x - x_{goal})^2 + (y - y_{goal})^2 + \sum_{n=0}^{N_{hazards}} \frac{1}{(x - x_{hazard_n})^2 + (y - y_{hazard_n})^2 + \frac{1}{f_n}} \tag{3}$$

This hazard generation function is very close to the one we desire—it is infinitely continuous over $\mathbb{R}$, can have an arbitrary number of hazard points, and will tend towards the goal as long as the boundaries are sufficiently far away from the goal. However, there is one further improvement that can be made: even with high fear, the area where the hazard is meaningful will be very small. This is because distance squared increases rapidly, and thus the reciprocal of the distance squared decreases rapidly away from the point. This will cause the solution curve to only narrowly avoid hitting the boundary. However, if the hazard analogy is taken more literally, such as where boundaries are other vehicles and a margin of safety is required, then it becomes critical to optionally increase the zone that the path avoids.

Figure 4: A Typical Hazard Function Viewed in Maple™



Plot generated from the function

$$H(x,y) = (x-2)^2 + (y-2)^2 + \frac{10}{(x+5)^2+(y+5)^2+\frac{1}{20}} + \frac{10}{(\frac{1}{5}(x+5))^2+(\frac{1}{5}(y-5))^2+\frac{1}{20}} + \frac{10}{(\frac{1}{10}(x-5))^2+(\frac{1}{10}(y-5))^2+\frac{1}{30}}$$

One common method to make functions have a larger area of effect is a horizontal stretch (See Appendix A). However, if a normal horizontal stretch is used, the boundary will change positions, as the center of the boundaries will not typically be at the origin. However, if the horizontal stretch includes the $x_{hazard_n}, y_{hazard_n}$, then the desired effect will be achieved. It is fairly intuitive to refer to the horizontal stretch factor as the caution size c of the boundary. Again, since it is more intuitive for caution size to increase as c increases, the reciprocal of c is used.

Finally, the numerator of the hazard fraction is defined to be 10, so that f is specifiable on a reasonable scale. The rationale: Currently the value of H is 100, just 10 units away from the goal location; boundaries should have a maximum of at least this magnitude. If the numerator is one, then fear has to be 100 to have that effect, but if the numerator is 10 then fear will have a more convenient range.

There are many more improvements that could be made to the hazard generation function. For example, if some of the boundaries move, then using extrapolation to predict future positions may be helpful. This would cause the hazard map to depend on time, but time dependence will not cause issues because time is the independent variable used later in the model anyway. Figure 4 shows a 3-dimensional rendering of a typical hazard function.

$$H(x,y) = (x - x_{goal})^2 + (y - y_{goal})^2 + \sum_{n=0}^{N_{hazards}} \frac{10}{(\frac{1}{c_n} \cdot (x - x_{hazard_n}))^2 + (\frac{1}{c_n} \cdot (y - y_{hazard_n}))^2 + \frac{1}{f_n}} \tag{4}$$

2.2 The Full Model

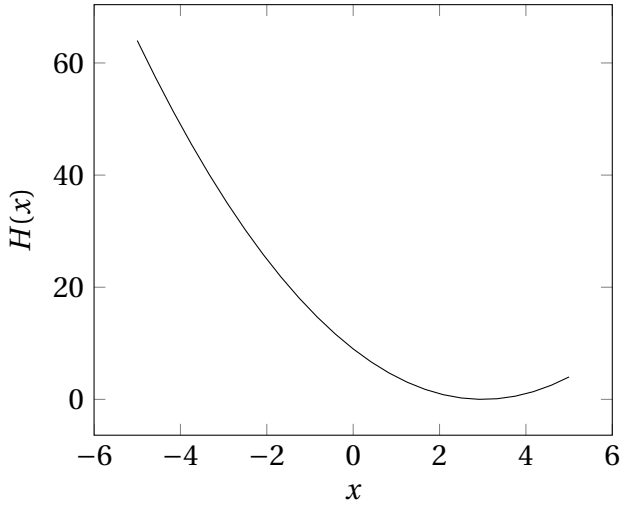
The goal of this derivation is to create a two-dimensional solution path by following the slope of H in $\mathbb{R}^3$. By pre-processing barriers in two dimensions, and assigning them values for fear and caution size, H can be generated.

Consider a simple, 1-dimensional example. Suppose that the path must start at $x = 0$, and must go to $x = 3$. Using the same default H where hazard increases as distance squared from the goal, the hazard function is defined to be

$$H(x) = (x - x_{goal})^2 = (x - 3)^2 \tag{5}$$

Consider a metaphorical ball that is placed at $x = 0$. The solution will follow the path the ball takes; namely, the **negative** instantaneous slope of H (see Figure 5). The slope at any

Figure 5: Simple Hazard Map for 1-Dimensional Problem



point is defined to be $\frac{dH}{dx}$. Note there is an element of time involved in the solution, so the rate of change of the solution path will be represented as $\frac{dx}{dt}$. Also, instead of using acceleration at each point to generate the path— as one would expect when taking the gravitational force into account—it is sufficient to let the velocity of the ball be directly proportional to the slope of the hazard at that point. This will give us a first-order differential equation of the form

$$\frac{dx}{dt} = -k\frac{dH}{dx} \tag{6}$$

Where x is dependent on t, and is the function that will be solved for. However, notice that the velocity of the metaphorical ball that generates the solution path will not remain constant as time goes on. This will cause strange accelerations along the solution path. Ensuring that the velocity is a constant at each point is a matter of dividing by the magnitude of the varying velocity $\frac{dH}{dx}$, which gives:

$$\frac{dx}{dt} = -k\frac{\frac{dH}{dx}}{|\frac{dH}{dx}|} \tag{7}$$

Substituting the specific values for this problem, using $k = 1$,

$$\frac{dx}{dt} = -\frac{\frac{d}{dx}(x-3)^2}{|\frac{d}{dx}(x-3)^2|} \tag{8}$$

$$\frac{dx}{dt} = -\frac{2(x-3)}{|2(x-3)|} \tag{9}$$

Starting at $x \leq 3$ implies $|2(x-3)| = -2(x-3)$

$$\frac{dx}{dt} = -\frac{2(x-3)}{-2(x-3)} \tag{10}$$

$$\frac{dx}{dt} = 1 \text{ when } x < 3 \tag{11}$$

This solution is logically sound— the best way to go from $x = 0$ to $x = 3$ is to move in the positive x-direction. In this example the velocity is divided by its magnitude so that the outcome is easy to see analytically, but when solving two-dimensional problems, the solution will not be normalized.

Broadening the same analogy to two dimensions gets trickier, but the concepts remain the same. First, create the hazard map using the method in Section 2.1. Then, set both the x and y-velocities equal to the x and y-components of the slope of the hazard function. One way to express the slope of the hazard function is the gradient, which points in the direction of greatest increase. The ball moves *down* the hill, so it will move in the direction of greatest decrease—the direction of the negative gradient.

$$\left\langle \frac{dx}{dt}, \frac{dy}{dt} \right\rangle = -\nabla H(x, y) \tag{12}$$

Expanding this equation for both the x-velocity and the y-velocity gives the system of equations,

$$\begin{cases} \frac{dx}{dt} = -\frac{\partial H}{\partial x} \\ \frac{dy}{dt} = -\frac{\partial H}{\partial y} \end{cases} \tag{13}$$

The desired solution curve is defined by the set of parametric equations $\langle x(t), y(t) \rangle$. Note that an initial position (a, b) directly translates to the initial conditions

$$\begin{cases} x(0) = a \\ y(0) = b \end{cases} \tag{14}$$

In most cases the partial derivatives of the hazard function will be fairly complex, typically yielding a system of ordinary, non-linear, first-order differential equations, which must be solved numerically.

3 SOLUTION

3.1 Analytically Solving Cases with Trivial Hazard Functions

Now consider the case of the trivial hazard function with no boundaries, where the initial position (x_0, y_0) is $(10, -5)$.

$$\begin{cases} H(x,y) = x^2 + y^2 \\ \frac{dx}{dt} = -\frac{\partial H}{\partial x} = -2x \\ \frac{dy}{dt} = -\frac{\partial H}{\partial y} = -2y \\ x(0) = 10 \\ y(0) = -5 \end{cases} \tag{15}$$

The two equations are not dependent on each other in this trivial case, so they can be solved independently using separation of variables. Compactly,

$$\begin{aligned} \int \frac{1}{2x}\frac{dx}{dt}\,dt &= -\int dt \\ \frac{1}{2}\ln|2x| &= -t + c \\ x &= c_1 e^{-2t} \end{aligned}$$

See Figure 11 in Appendix B for more detail. The same process clearly yields $c_2 e^{-2t}$ for $y(t)$. Next, resolve the initial conditions.

$$10 = c_1 e^0 \tag{16}$$

$$c_1 = 10 \tag{17}$$

$$-5 = c_2 e^0 \tag{18}$$

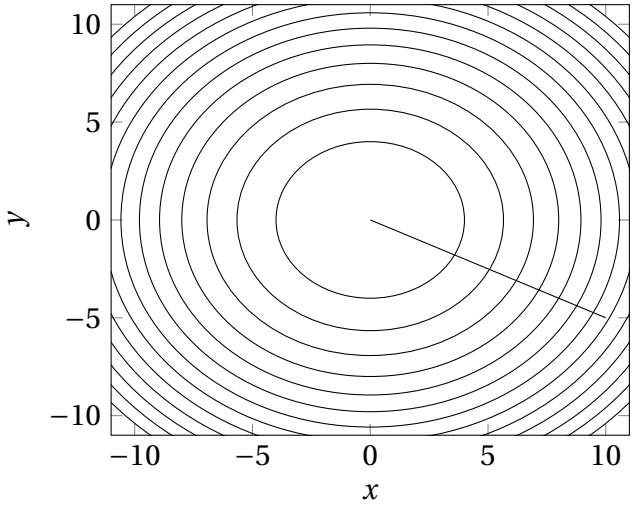
$$c_2 = -5 \tag{19}$$

The final solution is thus described by the parametric curve $\langle x(t), y(t)\rangle = \langle 10e^{-2t}, -5e^{-2t}\rangle$. Since $-5e^{-2t} = -\frac{1}{2}\cdot 10e^{-2t}$, the solution curve can be written simply in terms of y and x.

$$y = -\frac{1}{2}x \tag{20}$$

It is important to remember that the path ends when the metaphorical ball stops rolling, so the interval over which this function is valid ends when $\nabla H(x,y) = 0$. In Figure 6, we manually

Figure 6: Simple Solution with $H(x, y)$ Contours



plot the function on this interval, along with contours of H. Notice how the solution can be traced from the initial condition, $(x, y) = (10, -5)$, to the final position given in the hazard function by $(x, y) = (0, 0)$. Note that the interval restriction will come about naturally when solving numerically, because the solver won't be able to continue when it reaches $\nabla H(x, y) = 0$.

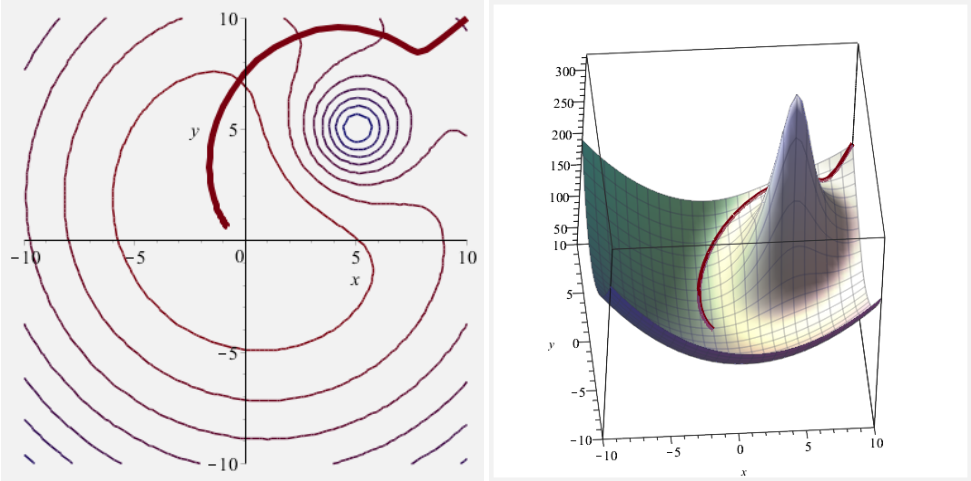
3.2 Solving Applicable Hazard Functions

Most useful systems that come about using this method are not currently possible to solve analytically. The method used to generate hazard functions results in a nonlinear system of differential equations if even a single hazard term is introduced, as seen below.

$$\begin{cases} H(x,y) = (x-1)^2 + (y-2)^2 + \frac{10}{(\frac{1}{10}\cdot(x-5))^2 + (\frac{1}{10}\cdot(y-5))^2 + \frac{1}{30}} \\ \frac{dx}{dt} = -\frac{\partial H}{\partial x} = -2(x-1) + \frac{x-5}{5(\frac{1}{100}(x-5)^2 + \frac{1}{100}(y-5)^2 + \frac{1}{30})^2} \\ \frac{dy}{dt} = -\frac{\partial H}{\partial y} = -2(y-2) + \frac{y-5}{5(\frac{1}{100}(x-5)^2 + \frac{1}{100}(y-5)^2 + \frac{1}{30})^2} \\ x(0) = 10 \\ y(0) = 10 \end{cases} \tag{21}$$

A numerical solver must be used to find solutions to nonlinear systems. Maple was used for this purpose, as well as for automating the generation of hazard functions. A Maple document that finds solution curves based on certain parameters is included in Section 3.4. $genHazard$—a convenience function that creates a hazard function—is defined within that

Figure 7: Contour and Projected Solution to (21)



document. Its signature is included for reference in Equation 22.

$$genHazard\left(\left[x_{center}, y_{center}\right], \left\{\left[x_{hazard_0}, y_{hazard_0}, c_0, f_0\right], ..., \left[x_{hazard_n}, y_{hazard_n}, c_n, f_n\right]\right\}\right) \tag{22}$$

As previously stated, f and c are the fear and caution size parameters respectively. $genHazard$ makes it very easy to explore the effects of placing hazards in arbitrary positions. Simply changing its parameters and re-executing the document is all that is needed to go through the entire process of generating a new hazard function and solution curve. For convenience, the document both superimposes the solution onto a contour plot of H, and projects the solution onto a 3-dimensional plot of H.

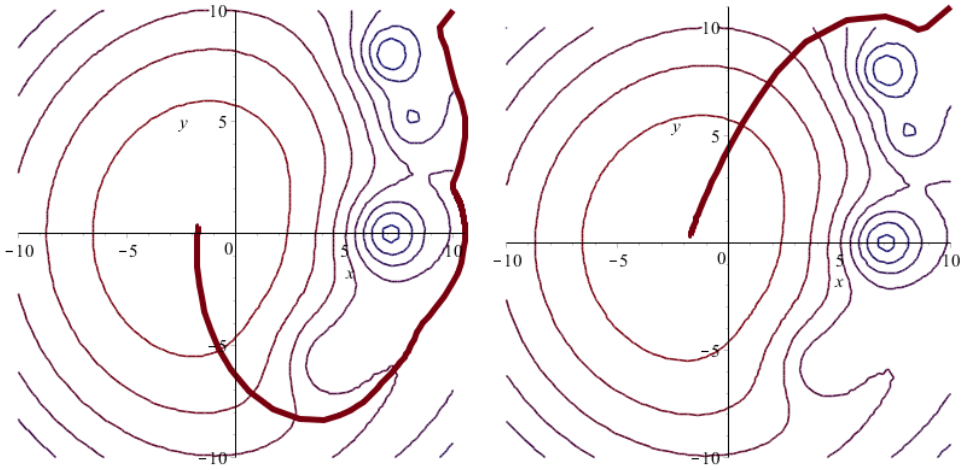
According to “Maple User Manual” (2005-2014), the default numerical solution method is "rkf45"—"Fehlberg fourth-fifth order Runge-Kutta method with degree four interpolant". This is the method that was used to generate the solution curves included in this paper. Figure 7 shows the numerical solution to the problem posed in Equation (21). The solution was obtained by using $genHazard\left([1,2], \{[5,5,10,30]]\}\right)$ when evaluating the Maple document included at the end of this section.

3.3 Limitations of the Model

Naturally, this model has limitations. As previously stated, the hazard near a barrier must be large and the hazard far from a barrier should be low. However, the hazard far from a barrier is never zero because it is not possible for a fraction with 10 in the numerator to be equal to zero; the fraction merely approaches zero as distance from the hazard point increases. As

a result, the position where $\nabla H = 0$ changes slightly whenever a hazard term is added, and there is no guarantee that the solution curve will arrive precisely at the final position that was initially specified. Although this result may not be ideal for applications requiring exact solutions, this imperfection is acceptable, perhaps preferable even, in the case of modeling human-like behavior.

Figure 8: Demonstration of an imperfect path

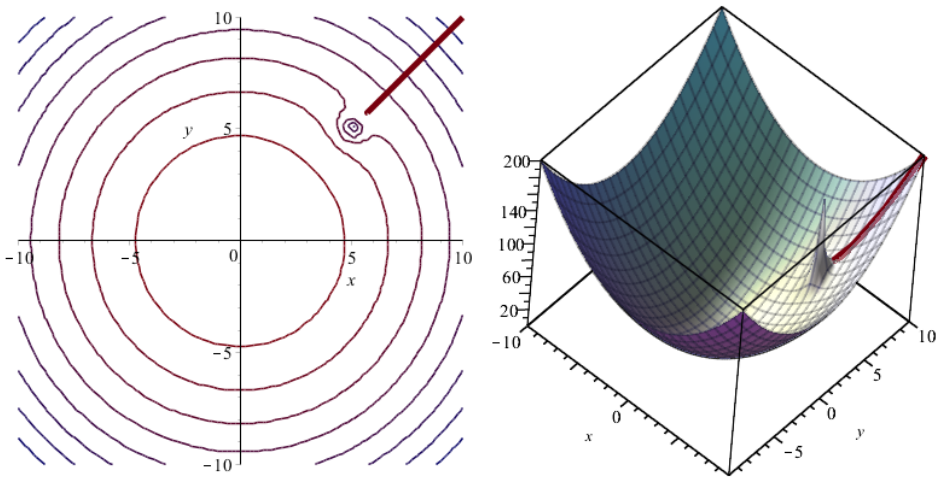


On a similar note, the solutions obtained may not necessarily describe an optimal path. The slightest change in an initial condition can cause a direct path to turn into a roundabout path. This sensitivty to small changes in initial conditions is relatively common in differential equations. In Figure 8, the initial condition $y(0)$ underwent a very small change, but the resulting path was much shorter. Thus, veering away from the destination near the start of the path could result in more efficiency in terms of distance traveled. Nonetheless, due to the nature of our model, moving against the hazard function is not an option, as the only initial conditions that we may define are the initial x and y values.

As mentioned before, simulation of human-like behavior does not necessarily require an optimal solution path; a path that looks vaguely indicative of intelligence may suffice, so choosing a non-optimal path based on a small difference in the initial position could potentially be useful in modeling actors for whom non-optimal paths are acceptable.

Complications also arise when the path arrives at a stable location that is not the intended destination. A simple example can be seen in Figure 9. In this case, the metaphorical ball is considered to be "stuck." Upon arriving at the intended destination, the ball remains in place because the goal is deliberately defined to be stable. However, there are other such stable locations where this behavior is unintended. This effect can be avoided by ensuring that the caution size c of most hazards is large enough to affect the path before it reaches an

Figure 9: An unintentionally stable location



unintentional point of equilibrium, and also by intentionally avoiding certain symmetries between hazard points and initial positions. In Figure 9, the caution size of the single hazard was very small and made no significant impact until the solution path came very close, *and* the initial position was defined such that the path would continue exactly to the equilibrium point.

3.4 Maple Document

$with(plots) : with(DETools) : with(plottools) : with(PDETools) :$

Each hazard point has [x,y,fear,cautionsize] where:
x,y - The center of the hazard point
fear - How afraid of the hazard you are (bigger -> more afraid. 20 average)
cautionsize - How much room to give the hazard (bigger -> more caution. 1 is average)

$$genHazard := (center, hazardPoints) \rightarrow \left((x, y) \rightarrow (x - center[1])^2 + (y - center[2])^2 + hazardTerm(x, y, hazardPoints) \right) :$$

(1)

$$hazardTerm := (x, y, hpts) \rightarrow 10 \Big/ \left(\left(\frac{1}{hpts[1][4]} \cdot (x - hpts[1][1]) \right)^2 + \left(\frac{1}{hpts[1][4]} \cdot (y - hpts[1][2]) \right)^2 + \frac{1}{hpts[1][3]} \right) + hazardTerm(x, y, hpts[2..-1]) :$$

$$hazardTerm(x, y, \{\}) := 0 :$$

Use the *genHazard* function to create a hazard function.

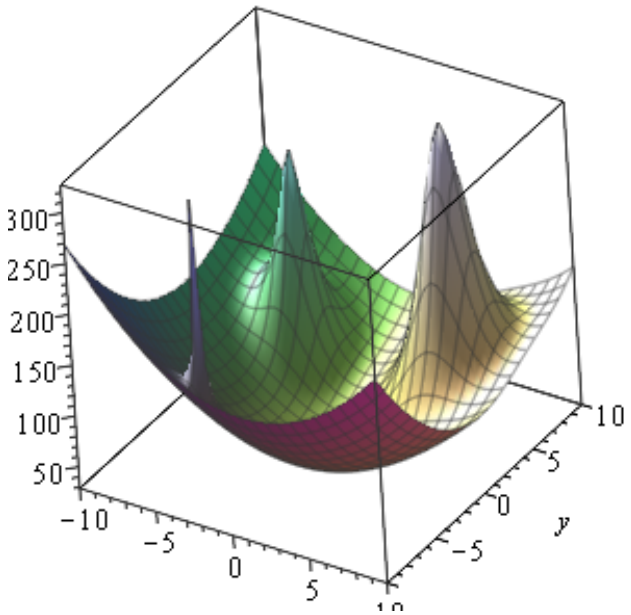
$H := \text{genHazard}([1, 2], \{[-5, -5, 20, 1], [-5, 5, 20, 5], [5, 5, 30, 10]\})$

$$(x, y) \rightarrow (x - [1, 2]_1)^2 + (y - [1, 2]_2)^2 + hazardTerm(x, y, \{[-5, -5, 20, 1], [-5, 5, 20, 5], [5, 5, 30, 10]\}) \quad \textbf{(2)}$$

$H(x, y)$

$$(x-1)^2 + (y-2)^2 + \frac{10}{(x+5)^2 + (y+5)^2 + \frac{1}{20}} + \frac{10}{\frac{1}{25}(x+5)^2 + \frac{1}{25}(y-5)^2 + \frac{1}{20}} + \frac{10}{\frac{1}{100}(x-5)^2 + \frac{1}{100}(y-5)^2 + \frac{1}{30}} \quad \textbf{(3)}$$

$with(plots) :$
$hplot := plot3d(H(x, y), x = -10..10, y = -10..10, axes = boxed) :$
$display(hplot)$



$$dxteqn := diff(x(t), t) = -subs(x = x(t), y = y(t), diff(H(x, y), x))$$

$$\frac{d}{dt} x(t) = -2\,x(t) + 2 + \frac{10\,(2\,x(t) + 10)}{\left((x(t) + 5)^2 + (y(t) + 5)^2 + \frac{1}{20}\right)^2} + \frac{10\left(\frac{2}{25}\,x(t) + \frac{2}{5}\right)}{\left(\frac{1}{25}\,(x(t) + 5)^2 + \frac{1}{25}\,(y(t) - 5)^2 + \frac{1}{20}\right)^2} + \frac{10\left(\frac{1}{50}\,x(t) - \frac{1}{10}\right)}{\left(\frac{1}{100}\,(x(t) - 5)^2 + \frac{1}{100}\,(y(t) - 5)^2 + \frac{1}{30}\right)^2} \quad \textbf{(2)}$$

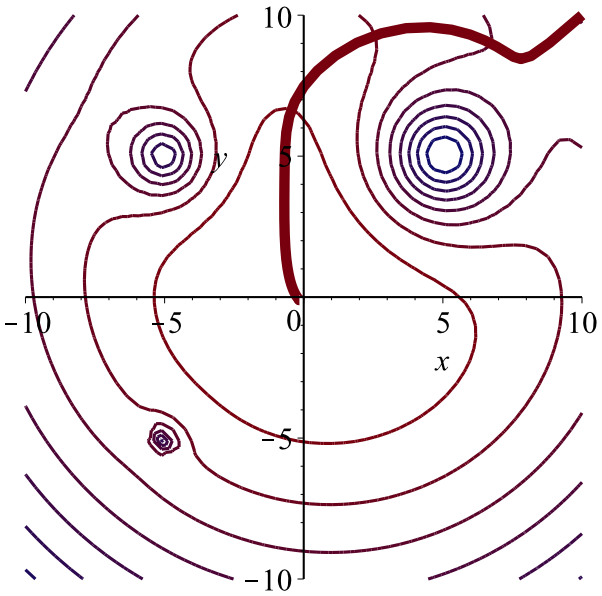
$$dyteqn := diff(y(t), t) = -subs(x = x(t), y = y(t), diff(H(x, y), y))$$

$$\frac{d}{dt} y(t) = -2\,y(t) + 4 + \frac{10\,(2\,y(t) + 10)}{\left((x(t) + 5)^2 + (y(t) + 5)^2 + \frac{1}{20}\right)^2} \quad \textbf{(3)}$$

$$+\frac{10\left(\frac{2}{25}\,y(t)-\frac{2}{5}\right)}{\left(\frac{1}{25}\,(x(t)+5)^2+\frac{1}{25}\,(y(t)-5)^2+\frac{1}{20}\right)^2}$$

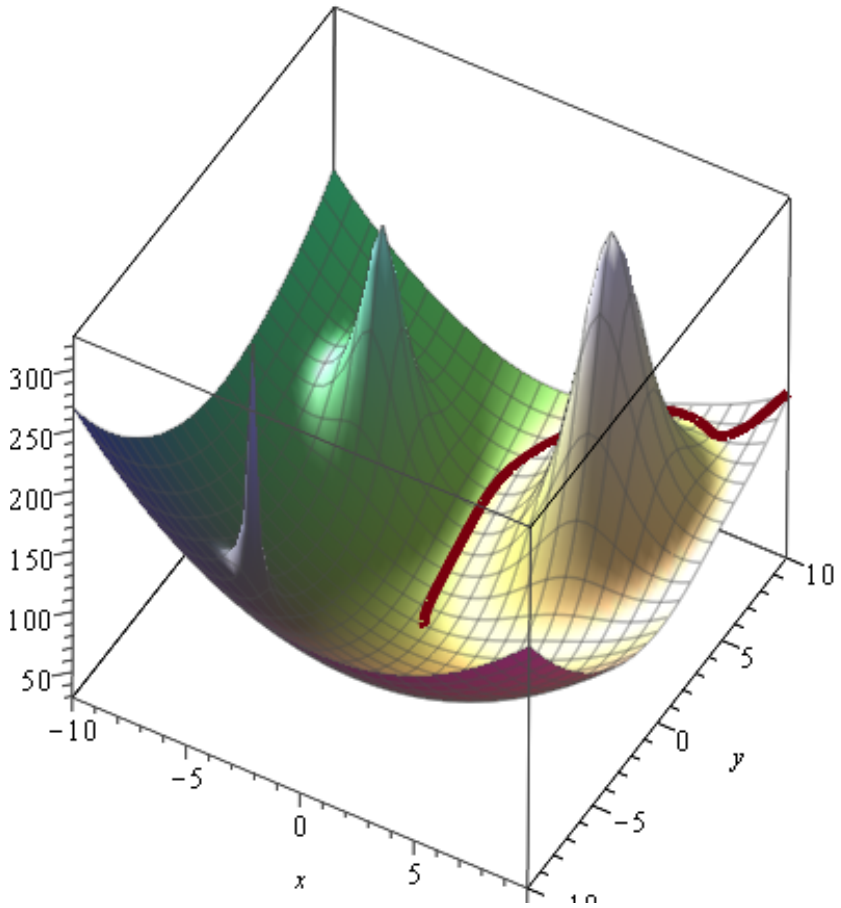
$$+\frac{10\left(\frac{1}{50}\,y(t)-\frac{1}{10}\right)}{\left(\frac{1}{100}\,(x(t)-5)^2+\frac{1}{100}\,(y(t)-5)^2+\frac{1}{30}\right)^2}$$

```
system1 := {dxteqn, dyteqn, x(0) = 10, y(0) = 10} :
soln := dsolve(system1, {x(t), y(t)}, numeric) :
solnplot := odeplot(soln, [x(t), y(t)], 0..1000, numpoints = 10000, thickness = 5) :
contourplotH := contourplot(H(x, y), x = -10..10, y = -10..10) :
display([contourplotH, solnplot])
```



For some reason, maple doesn't like evaluating H(a, b) directly, so this fixes that by forcing maple to treat H like a plain old function.

```
G := (x1, y1) → subs(x = x1, y = y1, H(x, y)) :
solnplot3DTransform := transform( (x, y) → [x, y, G(x, y) + 1]) :
display([solnplot3DTransform(solnplot), hplot])
```


300
250
200
150
100
50
−10
−5
0
5
x
10
5
0
−5
y

4 CONCLUSION

Pathfinding has traditionally been a difficult problem to solve both correctly and efficiently. A* is one example of a "pretty good" algorithm that finds an optimal path, consuming a somewhat reasonable amount of resources. However, there is still much room for improvement.

The model discussed in this paper doesn't necessarily produce an optimal solution, and its efficiency wasn't within the scope of this paper. However, it does seem promising. Due to its nature, finding a solution path analytically is nearly always impossible. As such, numerical methods were applied to solve several example differential equations, and the Maple program that was used in this process has been included. The solutions to these differential equations are curves that navigate through hazard fields. They are generated by following the path of least hazard at any point along the path.

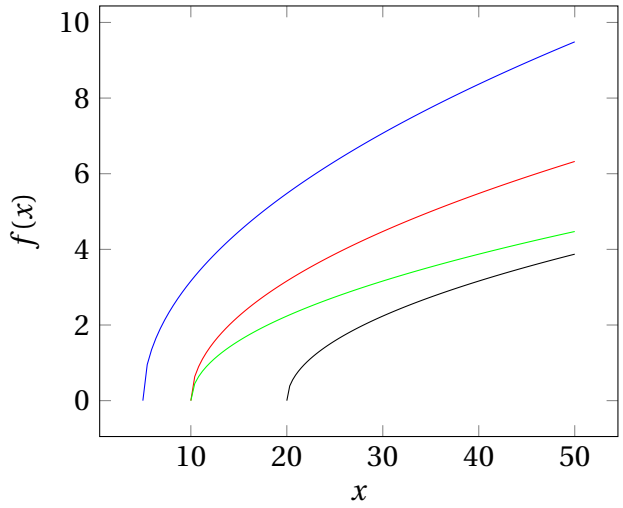
For practical applications, a numerical solution is not necessarily worse than a strictly analytical solution. Numerical methods are iterative by nature, and can be calculated on-the-fly as an actor travels through the path. This iterative method of solution, perhaps frame by frame in the case of video games, may actually be advantageous when compared to calculating a whole path at once using a traditional, grid-based pathfinding algorithm. By recalculating the hazard field in real time as the actor travels along the path, it is possible to account for changes in the path as they occur.

Unfortunately, this model does has several limitations, some of which include the possibility of stopping at accidentally stable points, arriving at a destination which is slightly incorrect, and the generation of inefficient paths caused by an inability to bear increased hazard temporarily for a reduction in hazard later along the path. However, some of these limitations may provide additional realism; people are not computers, and are as such not guaranteed to take a perfect path. The model introduced in this paper is thus useful for approximations of a best path as would be appropriate for video games and other non-critical simulations of human pathfinding, and less useful for vehicle routing or other problems where the possible set of blockades is statically known.

A HORIZONTAL STRETCH

A horizontal stretch is performed by multiplying the independent variable with a constant. When the constant is greater than 1, the function will compress, and when the constant is less than 1 the function will stretch. Constants that perform horizontal shifts must also be multiplied by the constant in order to prevent shifting the origin.

Figure 10: Red: $c = 1$ (Unstretched), Black: c=$\frac{1}{2}$ (stretched), Blue: $c = 2$ (compressed), Green: c=$\frac{1}{2}$ with shift also fixed



Base Function: $f(x) = \sqrt{x-10}$. Red: $f(x) = \sqrt{x-10}$, Black: $f(x) = \sqrt{\frac{x}{2}-10}$, Green: $f(x) = \sqrt{\frac{x}{2}-5}$, Blue: $f(x) = \sqrt{2x-10}$

B MISCELLANEOUS EQUATIONS

Figure 11: Using Seperation of Variables to Solve the Trivial Hazard Function Analytically

$$
\begin{aligned}
\frac{dx}{dt} &= -2x \\
\frac{1}{2x}\frac{dx}{dt} &= -1 \\
\int \frac{1}{2x}\frac{dx}{dt}\,dt &= -\int dt \\
\int \frac{1}{2x}\,dt &= -\int dt \\
\frac{1}{2}\ln|2x| &= -t + c \\
e^{\ln|2x|^{\frac{1}{2}}} &= e^{-t+c} \\
\sqrt{2x} &= e^{c}\cdot e^{-t} \\
2x &= \left(ce^{-t}\right)^2 \\
x &= \frac{c^2}{2}e^{-2t} \\
x &= c_1 e^{-2t}
\end{aligned}
$$

REFERENCES

Algfoor, Z. A., Sunar, M. S., & Kolivand, H. (2015). A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology, 2015*. doi:10.1155/2015/736138

Babel, L. (2012, August). Three-dimensional route planning for unmanned aerial vehicles in a risk environment. *Journal of Intelligent & Robotic Systems, 71*(2), 255–269. doi:10.1007/s10846-012-9773-7

Engebretsen, S. (2014). *A pde based approach to path finding in three dimensions: solving a path finding problem for an unmanned aerial vehicle* (Master's thesis, Norwegian University of Science and Technology).

Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in computer games. *ITB Journal, 8*, 57–81.

Lucas, S. M., Mateas, M., Preuss, M., Spronck, P., Togelius, J., Cowling, P. I., ... Bouzy, B., et al. (2013). *Artificial and computational intelligence in games*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik GmbH.

Maple user manual. (2005-2014). Maplesoft, a division of Waterloo Maple Inc. Retrieved from http://www.maplesoft.com/support/help/maple/view.aspx?path=dsolve/rkf45

Xu, X. (2014, October). Pathfinding.js visual demo. [Algorithm visualization]. Retrieved from https://qiao.github.io/PathFinding.js/visual/