Evolutionary Line-of-Sight Path Generation

Team 1

Max Benjamin Bond | Monte del Sol (Charter (Middle-High) School)

Arlo James Barnes | The MASTERS Program

Erik William Nelson | Academy for Technology and the Classics

William Christopher Fong | Academy for Technology and the Classics

Executive Summary

In telecommunications, it is often necessary to transmit a signal between two transmission points using a line-of-sight method of communication. Such communication generally requires a direct, unobstructed view from transmitter to receiver. However, in a complex terrain, resembling many real-world scenarios, obstacles may prevent a line-of-sight connection from being established, in which case it would become necessary to bounce the signal across several retransmission points. Such retransmission points can be expensive, however. The goal of our project is to create a genetic algorithm to find paths through such terrains in an efficient manner.

Problem Statement

In the communications section of operations science, it is sometimes necessary to connect two disparate points via a direct line of sight.

When no obstacles are present, such a transmission is trivial. In complex terrain, obstructions make the problem difficult, as a path must be built to move around them through the placement of relay stations.

An infinite number of paths are possible, however certain solutions will be more efficient than others in terms of number of relay stations and the length of the path. Calculating these superior paths is the crux of our problem.

Background

Obstructions are areas where retransmission points cannot be placed, and where the path cannot cross. These are equivalent to mountains, rivers, valleys, and other areas of the real world. Obstructions add complexity to the problem, since instead of a straight line stretching from two points, we must now construct a multi-legged line arching around obstacles.

An important concept to understand when solving any problem computationally is solution space. A problem's solution space is the set of all possible solutions that solve the problem.[4] The scope of a problem's solution space must be taken into account when designing an algorithm to solve it.

Modeling Parameters

The problem's complexity leads to a very large solution space, which is difficult to traverse deterministically. Therefore, we need to use a heuristic methodology.

Heuristics uses knowledge of a small portion of the solution space to navigate towards more optimal solutions.[5] In this way, it trends towards a perfect solution, even if it cannot be guaranteed to ever reach it.

One notable subset of heuristics is evolutionary algorithms. Evolutionary algorithms model nature's method of small, continuous improvement, moving towards a more optimal solution for its environment with each generation.[7] A subset of these, genetic algorithms, more specifically involves genomes.[6] Genomes are recipes for solutions that can be 'bred' together, or combined to produce 'children' with similar characteristics.

Genetic algorithms represent possible solutions as individuals in a population. Individuals are said to be 'feasible' only if they solve the problem[4]. Only viable individuals are allowed to breed. Genetic algorithms prune individuals who are less than optimal, based on a fitness function. The fitness function rewards those solutions who are more efficient, making them more likely to pass on their genetic material. In this way, characteristics beneficial to optimality are gradually selected for.

Solution Method Description

Our fitness function is based on the length of the path, and the number of relay points used. When solutions are bred together, genes are selected randomly from parents, but not fairly. The more fit parent has a higher probability of passing on their genes.

The length, relay points, and feasibility of a solution are calculated using Dijkstra's Algorithm for Shortest Path. Dijkstra's Algorithm, created by Dutch computer scientist Edsgar Dijkstra in 1954, is used to find the shortest path in a network with edge lengths from a given node to any other node.[1] It can also, more to the point for line-of-sight routes, find the shortest path across a graph between any two points, if such a path exists. To illustrate: a list would be made of the nodes one jump away from the starting node and their corresponding distances. The closest node on that list would then also have a list made, making sure to record the index of the secondly-aforementioned node; thus, a trackback list of edges can be made, and the distance between any two explored nodes easily calculated with simple addition. The shortest path will be followed until it either reaches the destination node or the end nodes, in which case subsequently longer paths will be followed to find the destination node.

To determine a relay point's neighbors, as required by Dijkstra's Algorithm, we must determine to which points are visible by line of sight. This can be done by drawing a line between the two points, and testing if it collides with any obstacles. Any line originating passing through a rectangle, originating from the outside, must intersect with one or both of its diagonals, as they stretch fully across the rectangle's width and height. Therefore, we can simplify the problem of intersecting with a rectangle to two instances of the line intersection problem.

The line intersection problem can be solved by examining an angle's handedness. Lines that intersect will form both right and left handed angles, whereas lines that do not intersect will form only left handed or only right handed angles.[2]

At the beginning of each generation, we iterate over the population of solutions to examine their fitness. Those that are less than the average fitness plus one are culled from the population. For every individual eliminated, another is created. If two or more solutions survive, the deficit is filled by breeding solutions together. If less than two, new individuals are generated.

Next Steps

The code for this project, while mostly functional, is not yet sufficient to draw usable conclusions. We also plan to incorporate additional evolutionary concepts to account for weaknesses in the genetic implementation, such as simulated annealing, a process that mimics the heat tempering of cooling metal. Using simulated annealing with solution found by our genetic algorithm would allow for a finer grained search of the solution space.

Although the two-dimensional model is analogous to real-world retransmitter placement, it will be extended to provide more useful information. Adding features such as a third dimension, or weighting areas of the terrain as more or less preferable, rather than acceptable or unacceptable, would add depth to our project.

Acknowledgments

Nick Bennett, for being a snazzy, fabulous, neat, nifty, stylish and classy mentor;
The Challenge Consult group, Celia Einhorn, David Kratzer, Drew Einhorn, and Talaya White
Nico Marrero for reviewing our Project Proposal
and Bob Robey for reviewing our Interim;
and Edsgar Dijkstra for being so awesome.

Works Cited

[1] Cormen, Thomas H., Charles Eric. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Third ed. Cambridge (Mass.): MIT, 2009. Print.

[2] Bennett, Nicholas. *Identifying Turn Direction for a Path Through Three Points*. First ed. Santa Fe (N.M.): Adobe Acrobat, 2010. PDF.

[3] Robert Sedgewick, *Algorithms*, 2nd ed. New York: Addison-Wesley, 1989, pp. 350-351.

[4] Many. "Candidate Solution." *Wikipedia, the Free Encyclopedia*. Web. <http://en.wikipedia.org/wiki/Candidate_solution>.

[5] Many. "Heuristic." *Wikipedia, the Free Encyclopedia*. Web. <http://en.wikipedia.org/wiki/Heuristic>.

[6] Many. "Genetic algorithm." *Wikipedia, the Free Encyclopedia*. Web. 06 Apr. 2011. <http://en.wikipedia.org/wiki/Genetic_algorithm >.

[7] Many. "Evolutionary algorithm." *Wikipedia, the Free Encyclopedia*. Web. 06 Apr. 2011. <http://en.wikipedia.org/wiki/Evolutionary_algorithm >.