Autobahns from Scratch: Evolving Transport Networks

Leland Williams
St Cloud State University
St Cloud, MN

ABSTRACT

Minimum Spanning Tree algorithms have a superlative range of applications, but do not do well in devising transportation networks. They do provide a baseline for comparisons. An multi-objective function that navigates a trade-off between usability and cost is proposed, and a genetic algorithm that evolves networks to optimize their objective function result are introduced. The success of the objective function and the genetic algorithm is demonstrated by displaying the results. This approach is general to any optimization problem on connected graphs where cycles are desired.

KEYWORDS

network optimization, routing costs, spanning trees, genetic algorithms, transportation

1 INTRODUCTION

Minimum Spanning Tree (MST) algorithms are a workhorse in network optimization, for good reason. For graphs with distinct edge weights, they find the unique set of edges to connect all vertices at the least sum of edge weights quickly. Time Complexity is $O(|E|\log(|V|))$ for a graph $G = (V, E)$. Finding the MST of a graph has applications in a wide variety of network optimization problems, including designing circuit boards [Cormen et al. 2009] or planning a trip through a transportation network. However, to develop an optimal transport network, we need additional methodology. This can be accomplished by a Genetic Algorithm.

Ideal transport networks quickly connect goods and passengers from any point to any other. They are also expensive. Thus no government is likely to remove and replace an existing network, but when extending a network or planning a network for a new mode of transportation, designing a delivery 'trunk route' between distribution centers, or playing certain video games, an algorithm to optimize such a network could prove beneficial. It may also be applicable to other domains involving optimal connected (cyclic) graphs.

Figure 1 shows a transportation network derived by an MST algorithm (throughout this work we used Prims'). The edge weights assigned are the distance between the cities calculated using the cities' latitude and longitude. (Of course, real world costs involve additional costs for water crossings and other obstacles, but calculating such weights requires numerous concerns beyond the scope of this paper. A naive edge weighting suffices for proof of concept.)

Autobahns from Scratch, December 14, 2018, St Cloud, MN
© 2019 Copyright held by the owner/author(s).
ACM ISBN .
https://doi.org/

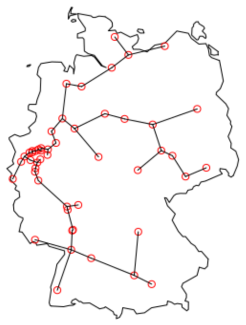

Figure 1: The Fifty Largest Cities in Germany, Connected by a Minimum Spanning Tree

Certainly investors or voters would reject a proposal to build this network . Some cities that are fairly near to each other 'as the crow flies,' particularly in the center, have a very long travel distance in this routing, and the largest cities, Berlin and Munich, have a very circuitous route to each other.

While this network is least cost, it is not the most optimal.

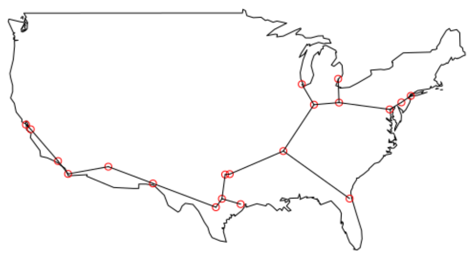

Figure 2: The Twenty Largest Cities in the United States, Connected by a Minimum Spanning Tree

Figure 2 shows the Twenty Largest Cities in the United States, also connected by a Minimum Spanning Tree. This set of edges would find approval from users based in Dallas and Fort Worth, but travellers between New York and San Francisco would not be happy with having to turn south from the Midwest to the Southern Border, then once at the Pacific travelling back up north to their destination. Again, while this is the minimal cost route, many users would consider having a network with a more direct East Coast - West Coast connection to be more optimal.

1.1 A Objective Function for Evaluating Transport Networks

Any algorithmic method to determine an optimal set of edges in a connected graph requires a method of comparison to enumerate their goodness. The objective function proposed here is multi-objective, as it considers trade-offs between two interest groups.

The first group of interest are the network's consumers - those who would ship their packages or themselves via it. These consumers are located in all cities assigned to the network, and have destinations in all possible cities. This suggests the use of an all points shortest paths algorithm. This algorithm takes as input a matrix of the edge weights between any two points in a proposed network. Entries on the main diagonal are zeros, and entries between vertices that do not share an edge are set to infinity. The output is a matrix of the weights of the least-cost path between each two points.

Once we have this matrix of shortest paths, we sum every entry to obtain the usage costs of a proposed network, and call it the APSP_SUM. In economic terms, the APSP_SUM is the negative utility of the system. In human terms, this is the 'pain' endured in using the network. Network consumers are motivated to lower it.

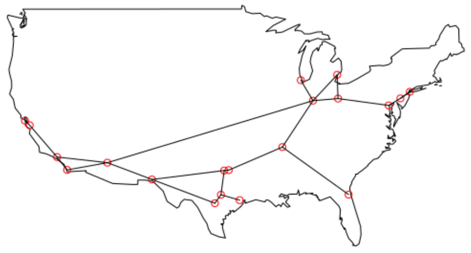

Figure 3: The twenty largest cities in the United States, connected by a network created by a genetic algorithm.

Refer again to the network in figure 2. The APSP_SUM of this network is 846490, the total distance in km of all possible paths, equally weighted. Compare this with the network in figure 3, a result of the genetic algorithm described below. It has some additional edges which offer more direct travel which lower its APSP_SUM to 764709. This is a nearly 10% improvement in the network usage cost and could allow improved supply chains for shipped components, or less dead time for commuting humans. Either represents a potential improvement in quality of life.

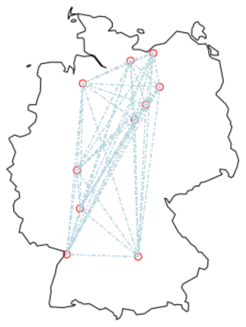

Figure 4: A complete graph of ten random points

Obviously, the lowest possible APSP_SUM is obtainable by using a complete graph as our proposed network, for example the one shown in figure 4. However, such a network is inefficient: paths between some cities could be easily routed through other cities, resulting in a minuscule increase in APSP_SUM. This network is also expensive: its construction costs may be more than investors or governments can afford, or at least are willing to pay.

Thus the second group of interest is the group that has to pay for the network to be created. This is the cost of the network, and is the sum of the weights of all edges in the network as determined by some weight function. For this paper, we use the distance between cities as the edge weight.

The proposed objective function optimizes a trade-off between cost and benefit. Again, we measure benefit as the difference between the APSP_SUM of a network derived by an MST algorithm and the APSP_SUM of a specified candidate network over the same vertices with the same weight function. This sum is then divided by the proportional increase in cost between the candidate network and the MST-derived network, similar caveats.

Formally, applying the APSP algorithm to the edge set E of a given graph to obtain the APSP matrix we express as APSP(E). The APSP_SUM formula is then:

$$\text{APSP\_SUM}(E) = \sum_{i=1}^{n} \sum_{j=i}^{n} \text{APSP}(E)_{i,j} \tag{1}$$

The cost function COST(E) evaluates the cost of a network by summing the weights of each edge included in an edge set.

$$\text{COST}(E) = \sum_{e \in E} weight(e) \tag{2}$$

Using both equations together we have our objective function:

$$\text{objective}(E) = (\text{APSP-SUM}(MST) - \text{APSP-SUM}(E)) \cdot \frac{\text{COST}(MST)}{\text{COST}(E)} \tag{3}$$

1.2 Two Observations

The objective function, when fed an edge list determined by an MST algorithm, will return zero (properly). Intuitively, a MST transport network is zero times better than itself.

If a graph is unconnected, the APSP will return a matrix containing entries of infinity, showing that since you can't get there from here, the travel distance is to there is, in a sense, infinite. On that edge set, the objective function will return minus infinity. Intuitively, an unconnected network is infinity worse than any spanning network.

2 GENETIC ALGORITHMS FOR GRAPH OPTIMIZATIONS

2.1 Genetic Algorithms

Genetic Algorithm are a class of stochastic solution search heuristics inspired by and in partial imitation of Darwinian Natural Selection [Davis 991a]. Typically a solution is a vector of setting, called a 'chromosome'. Unlike other vector optimization methods such like Hill Climbing or Stochastic Gradient Descent which iteratively adjust the entries of a single vector until a reasonable solution is found, a genetic algorithm maintains a (multi-)set or 'population' of candidate solution vectors which to improve their fitness in order to survive and have the opportunity to parent the next set of solutions.

Genetic algorithms, like single-vector methods, improve themselves by modifying the entries of their vectors, they also form new solutions by combining (usually) two chromosomes into a new one, a process called 'crossover'.

2.2 A Genetic Algorithm for Optimizing Transport Networks

The algorithm proposed herein is somewhat different than the typical ones, but it does follow the usual method of continually updating the population of chromosomes either a certain number of times, or stopping once the algorithm converges.

Convergence is determined by taking the standard deviation of the fitness, here measured by the objective function above, and exiting if said standard deviation falls below a certain threshold. This threshold was determined experimentally, as different problem sets would have different magnitudes of fitnesses.

The algorithm took as inputs the set of cities (vertices) and a matrix of edge weights.

2.2.1 Representation.

The algorithm begins by creating randomly creating an initial population. The chromosomes in this population are simply lists of edges that constitute a proposed network.

This differs from the more typical genetic algorithm variants because the order of edges in the list does not matter. Also, since the number of edges can vary from network to network, so too can the chromosome can vary in length from one to another, and the evolution of a solution can involve a change in length.

An earlier attempt at the problem used a degree-list: Each entry in the chromosome represented the degree of each vertex. However decoding the chromosome back into the same structure from which it was encoded proved difficult. If the least cost available edge was added first, then the longer routes were extinguished. In order to use this encoding it seems that it would have to be used with some sort of vertex weighting that would add some priority to which vertices a vertex might most wish to be connected to.

Anther possible, and perhaps more natural, representation would be that of matrix 'unfolded' into a list, with all entries zero in the absence of an edge between vertices, one in the presence of an edge. This approach was rejected because of the extra difficulty in maintaining a matrix for an undirected graph, where the constraint $M_{i,j} = M_{j,i}$ must be observed. Crossover would be particularly complicated. The possible correction of representing the graph as an upper-triangular matrix was also rejected for the extra difficulty in programming.

Thus the edge list representation was chosen since it was overall the simplest method.

2.2.2 Initialization and Population Size.

The algorithms used a fixed size population of twice the number of cities.

Algorithm 1: Random Spanning Tree algorithm

```
Result: Returns a new random spanning tree
Edges <- List();
Visited <- List();
Lonely <- cities.copy();
start_city <- random.choice(Lonely);
Visited.append(start_city);
Lonely.remove(start_city);
while Lonely.size() > 0 do
    s <- random.choice(Visited);
    t <- random.choice(Lonely);
    Edges.append(edge(s,t));
    Visited.append(t);
    Lonely.remove(t);
end
return Edges;
```

In order to 'seed' the algorithm with viable solutions the initial population consists of random spanning trees devised by the Random Spanning Tree algorithm, plus a copy of the minimum spanning tree, in order to add some 'obvious' edges into the initial gene pool.

If the algorithm began with a set of chromosomes containing a random number of randomly selected edges, the time to a just finding a spanning network could be very long, and the region of the solution space explored by the algorithm may be very small as the first chromosome to achieve connectedness have such dominance that the population would quickly converge around it.

2.2.3 Fitness.

The fitness scheme the algorithm used for determining survivors and parents is the objective function detailed earlier.

Chromosomes that did not encode connected graphs keep the score of minus infinity. While these graphs may encode useful partial solutions, there is no use in trying to assign 'partial' credit to them. Because the scoring varies from map to map, finding a scoring floor would be complicated. And, since the algorithm begins with all connected graphs and uses elitism in survival selection, no unconnected graph will ever survive to the next generation, so the question of scoring unconnected graphs is moot.

2.2.4 *Operators, and survivor selection.*

The algorithm used a generational approach, in which chromosomes of the current population were used to generate a set of new children of size equal to one-third the total population. The highest fitness members of the union of these sets form the population for the next generation.

In order to form a new chromosome, the algorithm chooses one chromosome from the existing population by tournament selection. The number of chromosomes that are chosen to participate in the tournament is one-tenth the number of vertices in the vertex set. The winning chromosome will act as the parent, the basic material for the new chromosome.

The parent has a 70% chance of being combined by crossover with another chromosome, also chosen by a similar tournament, and a 70% chance of being mutated. If both occur, the outcome of crossover is then mutated. Since both chances are evaluated separately, there is, of course, a 49% change that both crossover and mutation will be performed, and a 9% chance that the parent is copied into the new child set unscathed.

2.2.5 *Crossover.*

Since we have variable length chromosomes with edges in arbitrary order, a position-based crossover operator is infeasible. So for this algorithm crossover is performed by iterating over the edges in each list. For each edge in a list, if it is also in the other list, it is added to the new chromosome. (We observer that edge $(s, t) \equiv (t, s)$ so we search for both variants). Each edge we find that is not in both lists has a 45% chance of being added to the new list.

Algorithm 2: Crossover algorithm

```
Input: two edge lists, e1 and e2
Result: Returns a new chromosome from two given ones
child <- List();
for edge in e1 do
    if edge in e2 then
        child.append(edge);
    else if rand() < 0.45 then
        child.append(edge);
end
for edge in e2 do
    if edge in e1 then
        child.append(edge);
    else if rand() < 0.45 then
        child.append(edge);
end
return child;
```

In theory, you might search a wider space if the children grew and contracted in size further and faster, but it would be hard to control when to grow a short chromosome and when to curtail a longer one.

2.2.6 *Mutation Operators.*

The probability of 45% that an unshared edge would be included in the child chromosome means that chromosomes were more likely to shrink than grow during crossover. The mutation operator is more likely to grow the chromosome.

Three mutation variants were devised for use on this representation scheme. Once a chromosome is assigned to mutation, one of the three is chosen by weighted random selection and applied to the chromosome.

Twenty percent of the time a deletion mutation function is chosen. It randomly selects an edge and removes it from the edge set.

Thirty percent of mutations, an add mutation is used, where a randomly generated edge not already appearing in the edge list is added to it.

Fifty percent of the time, it will use a 'split' mutation. An edge (s, t) is randomly selected from the parent edge list and removed from it. A new vertex v is randomly chosen from the vertex list and the edges (s, v) and (v, t) are added to the edge list. The choice of the new vertex v is subject to the constraint that neither the edge (s, v) nor the edge (v, t) nor their equivalent edges (v, s) and (t, v) are already in the edge list.

3 RESULTS

The algorithm was applied to maps containing the fifty largest cities of France, Germany, Japan, Australia, and the United States. Twenty five attempts were made on each map, and each attempt resulting in success: that is, networks with a high positive fitness were produced. At the end of each attempt, if the best chromosome of the run had a higher fitness than those from earlier runs, it was saved. Maps of the highest fitness chromosomes produced follow.

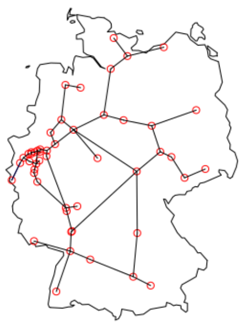

Figure 5: The fifty largest cities of Germany, connected by a network created by the genetic algorithm.

In Figure 5, in contrast to Figure 1 there are several more links through the center of the country, and there is a more direct path from Berlin to Munich.

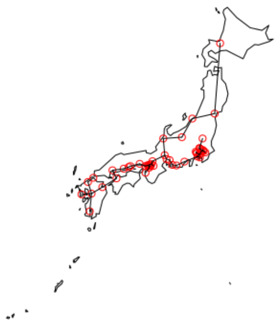

Figure 6: The fifty largest cities of Japan, connected by a minimum spanning tree

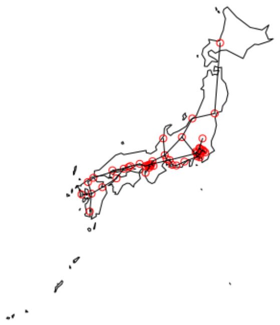

Figure 7: The fifty largest cities of Japan, connected by a network created by the genetic algorithm.

Comparing Figure 6 and Figure 7, we see that the genetic algorithm changed a city on the northern coast of Honshu to be a leaf, and added a second link through the bottom of Honshu.

Figure 8 is an example of a network produced by a minimum spanning tree that would never be constructed. It features several spiraline routes eminating from the capital region. In Figure 9 we see the genetic algorithm placed several parallel north-south routes, cleaned up an eastward route from the capital, split the northbound bound route from Paris into two routes.

Comparing Figure 11 showing the algorithmicaly devised network in Britain, we see, in similar contrast as in France, the placement of several north-south routes. Most notably, the link connecting to Belfast now originates in the Midlands instead of from Scotland, connecting Belfast more to the center of the map resulting in lower connection distances to each possible destination.

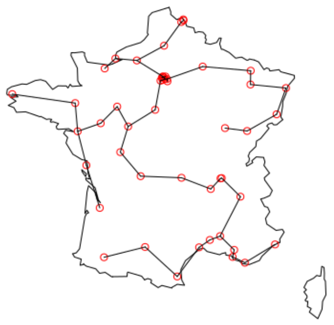

Figure 8: The fifty largest cities of France, connected by a minimum spanning tree

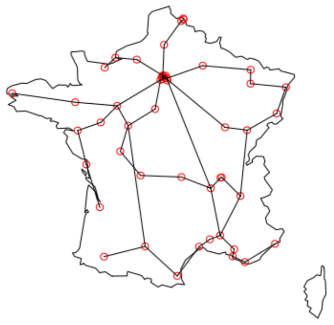

Figure 9: The fifty largest cities of France, connected by a network created by the genetic algorithm.

4 DISCUSSION

Even with the objective function firmly established, it is still easier to observe the success of the algorithm by the maps of its output. And we do see that the genetic algorithm produces networks far more usable than the minimum spanning tree; showing the viability of both the objective function and the design of the genetic algorithm.

Despite producing poor results on its own, the minimum spanning tree algorithm is very useful for producing a baseline low cost-high APSP_SUM network, and as a source of some reasonable short edges. Randomly generated spanning trees are likely to include lots of unnecessary links between distant cities. This is easy for a human to discern, but the objective function and the genetic algorithm are able to guide the evolution of networks that include these 'obvious' links.

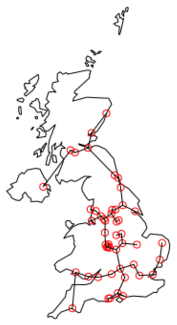

Figure 10: The fifty largest cities of the UK, connected by a minimum spanning tree

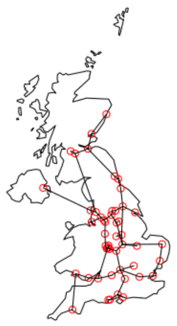

Figure 11: The fifty largest cities of the UK, connected by a network created by the genetic algorithm.

It can as well identify subtle combinations that provide better results for the network user as reasonable cost to the owner. Such combinations are difficult for humans to discern between, and laborious to produce. Thus the value of the approach.

As graphs are so very general, further applications are possible. Furthermore, the genetic algorithm is very flexible, and can accept several modifications. The weight 'function', actually a matrix, can be altered to meet different models of cost, and the construction cost matrix can be APSP_SUM can be calculated from a separate matrix, instead of using the same weight matrix for both as was done here.

One concern that can be accommodated by changes in the user's weight matrix is that of given preference to routes to and from the largest cities. This can be accomplished by altering the final APSP matrix by a factor of each cities percentage of the total population.

Another possible variation is constraining the network cost to fit a budgetary maximum. Chromosomes with cost greater than said maximum are either excluded from the population, or penalized to have a lower fitness. However the penalty method will likely result in the chromosome being not fit enough for inclusion in later generations, giving the same result but taking a longer time to reach it. likely.

REFERENCES

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition.* MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

Lawrence Davis. 1991a. What is a Genetic Algorithm? In *Handbook of Genetic Algorithms*, Lawrence Davis (Ed.). Van Nostrand Reinhold, Chapter 1, 1–22.