Genetic “road” building.

Southern Illinois University Dept. Computer Science, 2016

Advisor: Dr. Shahram Rahimi

Research/Software creation project.

Course: CS 437 (591 level)

**1. Abstract:**

The participants made a series of steps and programs which constitute a partial solution to the problem of designing the optimal paths (here “roads”) over an environment whose terrain is modeled as a grid of points and given two or more destinations to connect. The worth of the project, is found, to the degree it is found, in the phenomenon of people often selecting the places they want to place themselves and their activities (destinations) far apart and under the conditions imposed by the environment and then needing to construct the infrastructure to effectively travel between them over around and through obstacles. The participants used genetic algorithms in an attempt to solve this and an example is provided in item 6. While a research project in intent, it became more of a software development project, especially in light of the very light literature review. While the theory seems sound there are several limitations to the prevented process as listed in section 7.

**2. Statement of the problem:**

Given a 2D grid of points, which are used to approximately model an aerial view map and the terrain seen in it, build the optimum set of roads connecting any “destinations” (places people wish to come and go from), on that grid where the optimum roads are:

The, for all roads, cumulatively shortest set of roads (shorter roads are faster to traverse and less road is built thus less money is usually spent).

Are built through the areas that are least “costly” to build/drive through (what is “costly” is explained in the methods section).

Are specifically shortest for those roads stemming from the most frequently traveled to and from destinations (It is more important that a road which is likely frequently used be shorter than a road that is used infrequently).

The name “roads” is relative to scale. One could find the optimum highways between cities across a state, the optimum streets between city buildings, or the optimum sidewalks on a campus.

**3.Motivation:**

In many cases, especially at larger map scales, destinations (houses, cities, mines, airports, military bases, etc…) are not built without consideration of the environment in which they will be built. For example, an electric dam could be built in a field rather than next to a body of flowing water, but it would be less useful of a structure.

The destinations of our human world, the places we desire to go to and leave from, are often not close together and often their positions are decided on and/or found first, and then we have to connect them.

For example: Mines are built where ore and mineral veins are found, dry lake beds are great for building runways on for secret government aircraft testing facilities, due to a massive economic shift a city in china may spring up in a few months, cities are often placed near rivers or coasts to enable trade and travel, and places of geopolitical strategic importance are selected for forts and military outposts.

These examples are meant to illustrate that often but not always destinations are chosen or found first, often because of the value of the environment at the place. Then one desires to connect activities at one place to those elsewhere. Roads are perhaps the main means of powered transportation and one would like to minimize, as with everything, the costs of building them while making them as short as possible.

We attempted to do this with genetic algorithms.

As with most programs and problem solving, our “system” is a series of information machines, they and the flow of data through them, along with some human steps constitutes our attempted solution to this problem.

**4. Literature review:**

I have read no papers that deal directly with our topic. Most papers I found when I searched were for solving the next logical problem which is:

Given a set network of paths (“roads”) intersecting at nodes (“destinations”) which one can traverse to move between a starting node A and an ending node B, and one or move evaluations of fitness (usually the shortest path has a higher fitness), generate the best route from node A to node B.

This is the kind of problem well solved by the likes of Dijkstra's algorithm and indeed the important traveling salesman problem is an extension of this simpler problem, now with a succession of paths being optimized and a circuit as a constraint.

I do not say that there are no papers on our topic, only that I could not find any in the attempts I made to find them. Perhaps they were eclipsed by the related and more popular topic, perhaps I did not search hard enough.

A brief examination of the Michigan Department of Transportation’s article on how a road is built (1), while by no means exhaustive in its description of how roads are built, seemed to say that land availability and terrain type are considered in road design which makes sense. It seems irrational that such large scale, personnel laden and expensive operations would pay no heed to these things. However, I gathered that different road material types as well as other technologies enable one to build over land that may be less than ideal and modern earth moving equipment can allow one to warp environments to meet one’s needs. Given these abilities our model seems compatible with how roads are built, at least at this scale. Our model has certain areas that are illegal to build through, for example, land one does not have rights to build on, but more commonly there will be costly areas that one could build though if it was worth it, such as unfavorable soil types.

**5. Existing models and solution approaches:**

Again, I do not know of any existing approaches to this problem which use soft-computing because I do not know of any other attempts at solving this problem with genetic algorithms or other machine learning methods.

I do suspect, though this is not founded in any literature I have read, that given the near universal use of computers by governments and construction contractors, the two most commonly involved parties in building large scale roads, soft-computing decision assistance systems are likely or will in the future likely be used somewhere along the line.

**6. Proposed model and/or solution approach:**

Our partial, partial for reasons explained in the limitations section, solution is achieved using 8 steps and 5 (formerly 6) programs.

Programs:

*Grid maker* (Merged with *Road drawer*): Written in Python.

*Map maker*: Written in Python. (map\_makerV3.py)

*Path maker*: Written in Python. (path\_makerV3.py)

*Evolver*: Written in C. (ga-cpp.exe)

*Cycler:* A Batch script.

*Road drawer*: Written in Python. (size.py)

The steps are as follows:

Step 1.

The user finds an aerial view image containing the destinations and terrain they desire to model. It must be of .png file format to be compatible with a program in step 3.

Step 2.

The human choses max x and y values for the grid. The lower left corner of the image is always: 0, 0. The lower right corner is always: max x, 0. The upper left corner is always: 0, max y. The upper right corner is always: max x, max y. The greater max x and max y are the finer the grid and the greater “resolution” it can have in modeling the map.

For example:

If, for a very large map, we have a grid of only max x = 2, max y = 2, then we will have a total of 9 points each of which may have many miles, with regard to the map’s scale, between them. Since a destination’s location can only be approximated by a point on the grid, and so too with road obstacles like mountains, protected wildlife sanctuaries, swamps, etc… the closest point to any land formation we desire to model will be many miles off. The finer the grid, the closer our approximate point will be to its real world counterpart, in terms of relative position to the other modeled features of the environment, the environment in which one needs to build.

Also, the finer the grid the greater resolution we can model the shape of certain obstacles. If we have a very large grid, with few points, we may only be able to model a mountain as a square, things like this can lead to sub optimal solutions. Since it will raise a path’s cost to pass through the modeled mountain, and since the mountain is modeled as this box, the GA will be inclined to make paths that do not pass through the boxes corners. This is valid but not accurately representing of the mountain in real life, and building such a path in real life may result in a road that goes many unnecessary miles to avoid difficult terrain that is not actually there. The road will try to avoid a massive jutting corner of mountain that in reality is a set of smoothly sloping foothills that are easy to build through.

However, if we have a fine grid, we can model the mountain as a blob, with curves, and a more optimal path can be chosen that is close to the mountain, but does not pass through it, the curve approximating the mountain does not exclude so much area from being considered as cost free road space as the squares corner does. The higher the resolution of the grid the lower the disconnect between the model and reality. Though it should be noted that a higher resolution does come at greater computational expense.

Step 3.

The user uses the *Grid maker* program to draw a grid on the image. For exact instructions see Pouya’s readme file. Given the aerial view map and the max x and y values this will draw the corresponding grid on the map, where each intersection of lines or lines and edges are points on the mathematical model of the map. This grid lets the human approximate the position of the features of the landscape onto the grid. For example, with the grid drawn the human may see that the cities they wish to connect are approximated by points 4,5 and 17,32 respectively.

Step 4.

The user manually, for to do so automatically would require image recognition which we did not create, identifies the points which they consider to be of the following types:

Destinations:

These are single points or circular areas that are the places one desires to connect with roads, there must be at least 2. These have a priority, which is the user’s estimate of how important the destination is, that is to say how much traffic will come and go from it. For example, an airport will have a higher priority than a pizza shop by virtue of more people by at least two orders of magnitude, needing to go to and leave from an airport. Each destination has a name which is used to prevent a later program from drawing loops from destinations to themselves.

Cost “zones”:

N sided amorphous polygons where n >= 3. These are “bubbles” which are used to encapsulate and model areas that it is “costly” to build through.

Conditions can make land easier or harder, worse or better, to build on and/or drive through. Different ground types, legal restrictions, and gross terrain features such as flat plains vs mountains, vs large bodies of water, make certain areas easy, lest “costly”, to build through and others illegal or prohibitively expensive to build through. Additionally, considered costly are dangerous areas such as those where rockslides are likely or where visibility is quite commonly impaired or black ice is common.

For example, on a state wide level it would cost more to build a path over a mountain than across a plain, given the same straight line path. Not only will more energy be expended building the road over, or perhaps worse, through the mountain, but if it is built over the mountain the road will be longer than if it were on a plain, again, given that from an aerial view both roads appear straight. The cost zones can be nested and overlap, thus one can make the peak of a mountain more expensive than the foot of a mountain. There may also be zones one wishes to avoid for reasons that are not topographical, for example, it may cost one more money to build on land owned by one organization than it does to build a road on land owned by another. Each cost polygon has a cost associated with it, which is the used to calculate the additional cost incurred by entering a cost zone. While these costs are usually positive, that is to say one models places that are “bad” to go through. One can also assign a – cost, for a place one wants to go through.

For example: some land owners may pay you to build a road through their land because it offers the area business or lets them gain royalties on roadside ads.

Illegal zones;

The roads are constructed by connecting “active points” where an active point is some point on the grid. Which points are active varies from chromosome to chromosome. But the map doesn’t vary from chromosome to chromosome (and neither do the destinations or cost zones) and there are some places that are always illegal to build through. A partial solution is to ensure that all points in these circularly defined areas (which can be of any radius and can overlap) are always inactive, that is to say that the points in them may never be “active”.

For example, if there is an area heavily contaminated with radioactive waste, we do not want to have roads that pass through this, so we turn off all points in the grid that are in these areas. Or there may be a wildlife refuge one is not allowed to and does not want to build through, this would be defined as an illegal zone.

It is due to illegal and costly zones that this is a computationally complex task. If these could be ignored the correct answer would be much simpler, all roads would be practically completely straight and would either go directly from one destination to another, to a central hub or series of central/semi-central hubs, or some combination of these options.

The human manually compiles a list of x and y coordinates, and point indexes (the number of the point) where the index = (y\*(x max + 1)) + x.

Step 5.

One enters these into the *Map maker* program. For exact instructions see the readme file.

This program takes the indices (point numbers), max x and y values and x and y coordinates the human enters and fills several files which hold the mathematical grid the next programs can read (as opposed to the visual one the human can. It also makes the initial random chromosomes. It is at this stage that the user can enter the special points (destinations, illegal points, cost zones). Lastly a file is reset which records the chromosome which has had the lowest cost so far (blanked when this program is run because it is assumed we are starting a new map). The files are:

organism.txt: the repository for the chromosomes. It starts with the initially random chromosomes and is updated by a later program as evolution occurs.

map.txt: the repository of the grid of points, it records the index number, xy coordinates, status (D for destination, I for Illegal, nothing for standard point), and for destination points the priorities and names as well as if they are “independent” here meaning that they are allowed to grow a path. It is possible that one would desire a place to be a destination but not need it to grow its own road. For example, with only two destinations one only needs one road.

cycle.txt: blanked whenever *Map maker* is run, this holds the current cycle number (how many generations have passed), the best (least costly) chromosome so far and the line segments of all of its paths as well as the generation it was from.

cost\_poly.txt: The repository for all the cost polygons assigned to the grid currently in use.

In this program one also selects the number of chromosomes one wishes to work with and reasserts the max x and max y values that one started with, since this program does not “see” the grid one already drew.

An explanation of the grid as it relates to chromosomes is important at this point. Each map is reduced to an array of points starting at point 0 and ending at point x. This grid of x + 1 points is translated into a binary string where each such string is one chromosome and each individual digit is a gene, representing the “activeness” of one point for that chromosome. If a point’s digit is 1 that means that for that chromosome it is “on” and can be an ending and/or starting place for a line segment on a road. If it is 0 then the point is “off” for that chromosome and while paths can pass over it, they cannot land on it or spring from it. The roads for each chromosome are made by starting at points labeled as Destinations and successively linking them too active, “on” points, until a stopping condition is reached, as will be explained in the algorithm section below. For each chromosome, a random string is generated and all genes that represent illegal points are insured to be 0. The positions of these illegal points are recorded in the map.txt file that they may be passed along to the *Evolver* program so that it does not accidentally turn on, in its mutation and crossover, points that must remain off. Not every active point will necessarily be used in a path for a chromosome, but only active points can be considered to be part of the paths that are grown.

The general theory of application of a GA to this problem is that eventually, the chromosomes which have the combination of active and inactive points which lead to the lowest net cost for the roads generated from them will be selectively promoted to reproduce and their counterparts will be culled.

Step 6.

One then runs the *Path maker* program. This program requires no immediate human interaction other than running it, all its inputs come from files created by *Map* *maker*. This program draws in the data from the above files and constructs the roads for each chromosome and computes the cost of that chromosome from the cost of the roads.

The cost of a road = (the length \* the priority of that roads originating destination) + (the added costs)

For example, if a road is 10 units long and the priority of its originating destination is 20 then its baseline cost is 200 units.

The added costs for a road are the summation of all “optimistic” distances one’s road goes through each cost zone \* the cost of that cost zone. For example

If a road passes through two cost zones, one with a cost of 5 and one with a cost of 6, and it does so for distances of 3 and 15 units respectively, the added cost =

(5\*3) + (6\*15)

15+90

105

For this road the net cost would be 305.

I say optimistic because the *Path maker* program detects only collision with the perimeter of a cost zone and thus a path could enter a cost zone, hop between a great many active points inside of the cost zone, and then leave the cost zone, and the program can only approximate this by assuming the path takes the shortest (optimistic) route from entrance to exit. It was computationally easier to detect line collision than to detect point presence in the interior of an often irregular and amorphous polygon.

If this is the only road for a chromosome, then the chromosomes cost is 305. If the chromosome has more than one road, the other road’s costs must be calculated the same way and added as well.

Paths are draws by a simple algorithm that guaranties connection, but not optimal connection, optimization is achieved through the use of the genetic algorithm.

1. Establish the priority of all destinations:

Not all destinations must grow roads. But for those that do priority is established as follows:

Highest assigned priority, if there is a conflict (two or more destinations have the same priority) the we move to:

Closest to center. The reasoning here is that, if a destination is close to the center, it can act more like a hub for the modeled area and thus is more traveled through, if there is a conflict at this level (two or more destinations have the same distance from the center (unlikely but possible) we move to:

Lowest index, there can be no further conflict here as one point will have a lower index than another.

2. For each destination a road will grow from:

1.Grow head (current point the path is growing from) = Start point (the Destination’s point)

2.While the path is not finished growing

3.Grow to the nearest active point that is not yourself, that is not on the path that you are part of and that not is connected to a destination that is already connected to your start point (This prevents two destination loops (A -> B, B-> A)).

4.The newly grown to point is the new Grow head.

5.If the point you grew to is a destination or a path connected to a destination, you can stop growing, the path is finished.

6.End point = Grow head

This is guaranteed to create paths, though the system is somewhat imperfect as will be described. Paths are grown deterministically and the same chromosome under the same cost zone, illegal zone, grid and destination conditions will generate the same paths.

The cost is calculated as described and the chromosome with the lowest cost is stored in a separate file, cycle.txt, that it might be drawn when we wish to know the current best path.

It outputs all these costs and chromosomes to the output.txt, which is read by the next program in the chain.

Step 7.

The current chromosomes with their costs are given to the *Evolver* program, which commits the crossover, mutation and cloning. It outputs a new organism.txt file, containing the next generation of chromosomes, for the *Path maker* program which makes a new set of paths for the incoming chromosomes, and gives each a cost. The *Evolver* and *Path maker* programs pass data back and forth for as many cycles as the user desires to run them for. To assist in this running, *Cycler* was written, a small batch script program which runs this pair of larger programs for a set number of cycles without the user having to click one than the other.

Fitness is calculated from cost approximately as follows:

For all costs >= 0:

Fitness = 1/((cost/z) +1)

Where z is 1 to the power of 10 to the z. which is used to reduce the cost values, which can be quite large, by orders of magnitude. This is done because the Evolver program needs to transform costs into probabilities of parenthood and thus needs to normalize the incoming costs. See documentation for an explanation of this.

For all costs < 0:

Fitness = -cost

Step 8.

The Final step is the printing of the final paths. This is done by the *Road drawer* program which is explained in documentation of it. It is given the set of line segments of the best paths (it takes in a file named best.txt which is made by copying and pasting the 5th line and below of the cycle.txt file and the image the user started with. The image is given its grid and then the paths are drawn over that, thus one can see how the destinations connect by the most successful set of paths as of all cycles run.

**6. Results/examples:**

Below is the starting image of our example:

Max x: 19

Max y: 9

Number of points: 200



With its gird it looks like this:

It has the following measurements and “special points/zones”:

Illegal points, format is x,y,r where radius is the circular radius around the central point in which all points in the radius are also declared illegal:

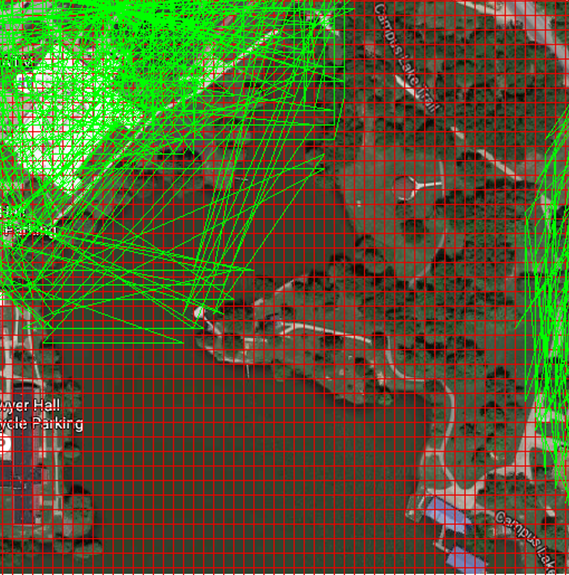
8,0,3

6,3,2

8,4,1

9,5,1

These are used to cover the points on or near the lake which is in this example illegal.

In an earlier attempt with 5000 points, we can see that while some paths do go over the lake, a success is achieved in that none end in the midpoints of the body of water, which were coved by illegal zone circles, only on the fringes which the circles missed:

Destination points:

Destination points can also have a radius allowing points to be considered part of a destination which are around a central point. This allows one to model destinations as amorphous shapes, such as a rough L shape or a large crescent, provided one describe the shape to *Map maker* with enough small and overlapping circles. This also allows paths to end on these edge points of a destination. However, in this case the radius of each destination is 0, meaning it is considered as a “point” destination.

Destination A, the main North side entrance to Engineering. Priority 5, x:18, y:3.

Destination B, the public policy buildings. Priority 4, x:15, y:7.

Destination C, Lentz hall. Priority 3, x:5, y:6.

Destination D, Pierce hall. Priority 2, x:3, y:4.

Destination E, Brown hall. Priority 1, x:1, y:4.

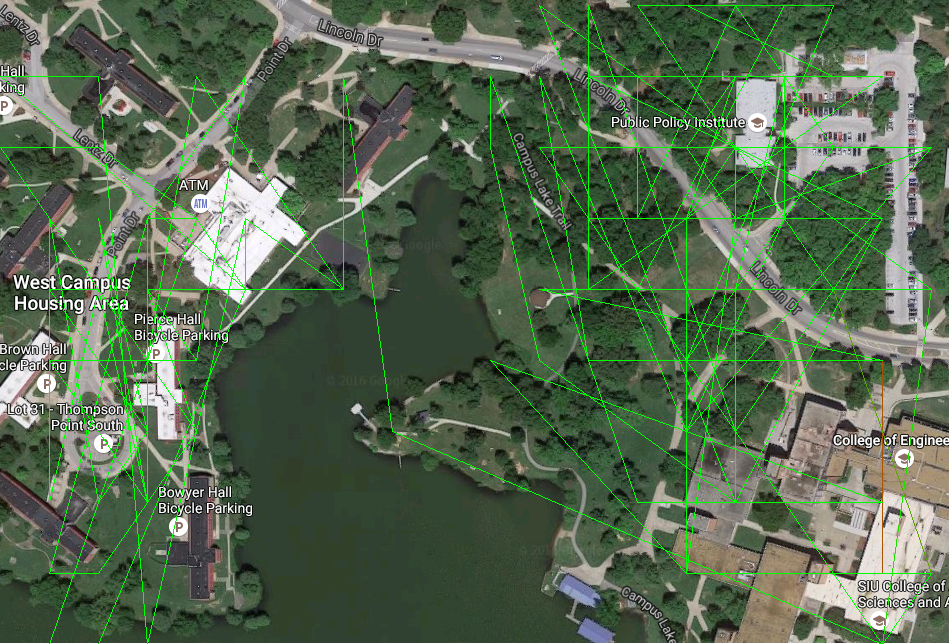
Cost zones:

Because Illegal zones prevent paths landing on certain points but not going over them a large cost zone is places over the lake to generate selective pressure against paths through the lake. Unlike the other two point types, cost zones are identified by their indices, the point numbers. The indices of the points are:

5, 84, 108, 149, 90, 68, 51, 13.

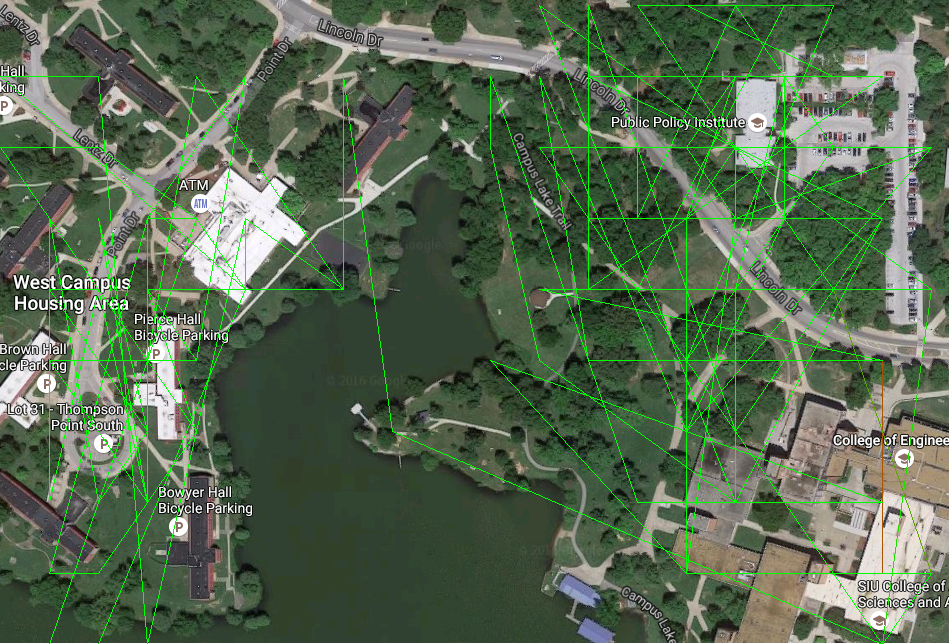
These indices create an 8-sided polygon around the lake with each indexed point being connected to the one before and after it. The first and last points are also connected to create the loop/bubble. I gave it a cost of 100.

Here is the first “best” path set, drawn on the image, this is before any evolution:



We can see that the lake is largely avoided and that the roads cluster around the 5 destinations. One road crosses the lake and as I think about it one might enjoy a stroll over the lake, on a bridge.

Here is the best path set after 500th generation, drawn:



…and they are virtually identical, only 2 line-segment differences, there were 113 lines in the 1st one and only 111 in this one…see limitations.

**7. Conclusions:**

Given my lack of knowledge as to other approaches to this problem, I cannot compare the success or failure of our method to them. Furthermore, while I can abstractly see use for our project for its intended purpose I do not know that it is that humanly worthwhile of a system.

A program that commits virtual origami folding might have use in protein folding analysis and thereby become an instrument for understanding the machinery of the cell, and enable the curing of many grave illnesses and degenerative conditions, as well as enable the creation of biological weapons whose results could be obscenely tortuous, completely fatal and in all practical terms unstoppable, a beautiful and terrible angel of death, combatable only but the same technology which gave it existence… I do not think our project touches a subject which is **that** important. But then, how am I to know? As far as I understand it, topological mathematics used to describe knots has found use in string theory.

I can say, at least for my own part, that having to write out a program to do something I find intuitively easy has demonstrated to me the amazing computational nature of our own minds and brains. I have known for some time that the brain is amazing, and have wondered at its ability, just as a cosmologist who feels awe and wonder at the majesty of the night sky in the desert or on some lonely cliff in the mountains. But the wonder does not stay forever, the day to day requirements on our attention call upon us, and we are once again unaware. But like a shooting star that draws the cosmologist’s eye one night, a brief moment of contemplation, as my program ran, of how amazingly powerful the brain is came again to me, and I enjoyed it. I remember a quote I read once:

“The computer is incredibly fast, accurate, and stupid. Man is unbelievably slow, inaccurate, and brilliant. The marriage of the two is a force beyond calculation.” – Leo Cherne

I do not have any journal level conclusions though. We started with no paths, our first path was pretty bad and our last best path was…nearly identical.

I value wonder and like that this project brought it to mind. And Cherne was right, we have been a powerful pair, I wonder what the next evolutions in computing mediums will bring, now that silicone and germanium seem to have run their course. That this project gave rise to these thoughts is not in itself a mark of the project’s quality and I include it as it is of personal value to myself rather than a significant empirical result for any reader.

I also think that this was a valuable project in that it educated me practically in the course, by letting me put my theory into practice.

**8. Limitations:**

There are of course some limitations and flaws that it is worth mentioning, both to show we are aware of them and to inform others that might, in an unlikely turn of events, pick up where we left off.

Perhaps the most notable limitation is that the instance we use of a genetic algorithm produces a great deal of homogeneity causing high cloning of the first generation’s lowest cost chromosome without much mutation. In our final example we used 50 chromosomes in 500 generations and still this 1st chromosome dominated with barley any changes. In fact, after counting I found 24 clones of this chromosome and 15 clones of another, meaning that even if all the others were different from these and each other there are still at most 13 chromosomes. This is some unhelpfully strong evolutionary favoritism, especially for a path that isn’t that objectively good. I suggest a higher mutation rate and a more careful normalization of cost values when computing fitness.

The *Path maker* allows only one road to grow from any one destination which is no how it should work. In real life destinations are allowed to have more than one road come in and out of them. Additionally, roads that chain destinations are not possible with the present programs. A valid road might pass through one destination and end at another, but this is not possible at present.

*Path maker* also cannot prevent paths from being drawn “over” illegal zones, only from landing in them, this makes it less likely but not impossible that roads can be drawn through illegal areas.

Also, it was originally intended that the multiple for priority that give a road its baseline cost be the average of the two priorities of the destinations connected. For example, the road 10 units long connecting the airport with a priority of 20 to the pizza shop with the priority of 3 would have a cost of 115, which is the length, 10, times the average priority, 11.5. But it does not do this.

I desired this because I felt it would more accurately model the way roads are actually prioritized, even if a road comes from an important place, if it goes to an unimportant place it will rarely be used and should have a lower priority.

It was also originally intended that if there is only one destination left to grow a road, and it is already connected, it would not grow a road, as it might well be superfluous. But it does not do this.

There is also the problem of having only two destinations. One needs a minimum of two destinations for the program to run and not crash, but can engineer circumstances where there are only one or two destinations and still have it crash, that is to say the program is not “smart” enough to prevent one from causing these things.

I did not find an easy way to detect length of path within a cost zone, and thus *path maker* can under estimate the cost of a path. For example:

If I have a large vertical rectangular cost zone, and a road enters at the zone’s mid-point and immediately leaves out the other side, it may have an added cost of 10. If another road enters at the same point, travels up and down many times inside the cost zone, and then leaves at the other side mid-point, it too will receive an added cost of 10. True the longer path will have a higher baseline cost, but the present program cannot detect distance of path within the cost zone, only between entrance and exit points.

The *Road* *drawer*/*Grid maker* program imperfectly fits the grid to the image, allowing some of the grid and thus its adherence to the intended special points on the map, to be lost.

Lastly, and this is a problem, some of the programs currently do not run on each of the three project participant’s machines, meaning they have to keep passing data back and forth rather than one person being able to gather all the programs and work on the whole system. only run on the machines they were created on. *Evolver* requires a specific C++ library be installed to my knowledge, *Road* *drawer*/*Grid maker* requires one install PIL (Python Imaging Library). Additionally, anyone wishing to run *Path maker, Map maker* or *Road* *drawer*/*Grid maker* needs to have the baseline python 3.5 or later installed on their computer, though we have found this easy to do for each of our machines.

1. http://www.michigan.gov/mdot/0,1607,7-151-9615-129011--,00.html