

Using Genetic Algorithms to Optimize Sign Allocation Along Otsiningo Park Greenway Trails

Joshua Brandoff

May 6th, 2008

Disclaimer: Portions of this paper use technical language directed more towards the academic community. If you are a casual reader and do not understand a section, simply skip to the next section and continue reading.

Binghamton Greenways and Trails

Of the four research areas offered as part of the CIC-LI curriculum, the “greenways” development project offers the greatest opportunities for rapid results. Spearheaded by Department of Transportation liaison Mark Bowers and Broome County Department of Planning and Economic Development associate Gail Domin, the project aims to make innovative use of open “green spaces” along such routes as I-84 and Route 434 and improve the usability of existing spaces (such as parks). CIC classmate David Miller and I have decided to focus on the development of creative and functional sign design and layout for use on the “Greenway” trails. Calling upon my background as a Bioengineer, I use what are called “genetic algorithms” (GAs) to find the optimal physical distribution for the signs along trailways. Genetic algorithms use the rules in the Theory of Evolution to “grow” solutions and compete them against each other to see which ones are the most fit (in this case, which distribution of signs best meet the optimization requirements). The signage distribution must be optimized in such a way that it is functional to as many local residents as possible. Thus, using an optimization technique like genetic algorithms appears to be the best course of action.

What are Genetic Algorithms?

The easiest way to describe genetic algorithms is to use their biological and evolutionary analogues, as reviewed in *Simulating Neural Networks* (Freeman, 1994). The structure that encodes the instructions on how an organism is built is called the *chromosome*. One or (as is often the case) more chromosomes are required to specify an entire organism. The complete set of chromosomes for an organism is called a *genotype* and the resulting organism itself is called a *phenotype*. Each chromosome is comprised of *genes*, where each gene encodes for a particular feature of the organism. The location of the gene within the chromosome, called the *allele*, determines what characteristics the gene is coding for.

Development of new genes involves sexual reproduction between two parent phenotypes, resulting in one child phenotype. During this mating process, the chromosomes of the parents are mixed together to form the chromosomes of the children. The idea of sexual reproduction is such that the *fitness* of the offspring increase over time. This means that the children acquire the best traits from the parents and are more likely to survive in a given environment.

Mechanics of Genetic Algorithms

In a computer simulation, genetic algorithms are usually represented by a string of some type. Here, a string is considered to be a series of characters (e.g. "AGTCAGACA"). Each position in this string is a gene which can take on different values. When used in conjunction with neural networks, these values are weights and biases leading into hidden layer and output neurons.

Fitness is quantified with commonly used (by engineers) diagnostics, such as the mean squared error (MSE) or A_z value. Other diagnostics used include ROC curves, partial A_z values, sensitivity, specificity, and positive-predictive value (PPV). Networks (chromosomes) with higher fitness scores are more likely to reproduce.

Thus, like an organismal population, a GA simulation begins with a population of networks generated randomly. The fitness of each network is then be determined, and these

fitness scores are used to select parents for the next generation. These parent networks would then "mate" and produce a child network for the next generation.

The details of the selection and mating processes are what make genetic algorithms different from other optimization tools. With GA's, parent selection is performed using a "roulette-wheel method," whereby each member of the population is given a piece of the wheel proportional to its squared fitness. This is also known as a directed random search.

Reproduction of the children involve *crossover* and *mutation* processes. In crossover, each parent provides part of the genetic information necessary to produce a child. The proportion of genes shared depends on a randomly chosen crossover point. This is called single-point crossover and is the process used for the GA's in this paper. It has been demonstrated that crossover can result in populations where the overall fitness increases over time. Because the crossover process might result in populations that converge at a suboptimal fitness, a small mutation factor is included where genes (or bits) in the chromosome are randomly switched. Because mutation is so random, its probability of occurrence is often kept on the order of .1%.

The overall process of renewing the given population also varies. The most general method, *generational replacement*, replaces the entire population after each generation. The problem with this method is that there is not any guarantee that the children will turn out better than the parents. In addition, the individuals (in this case, networks) with the highest fitness may be lost because they were temporarily at a lower fitness than other poor-performing networks.

This is countered by a process called *elitism* where the best individuals from the previous generation are guaranteed to be in the next generation. Elitism is the renewal process of choice used by many research groups (Bevilacqua, 2001) (Heckerling, 2003) and is the process used by the GA-NN simulations for this paper.

Genetic Algorithms as a Practical Tool

The use of genetic algorithms as a tool for industrial or civil engineering purposes is well documented; the academic community has done an excellent job of illustrating the practical uses of genetic algorithms. *Practical Genetic Algorithms* by Randy Haupt, there are several examples

of how genetic algorithms can be applied to community issues. One example describes optimal positioning of an emergency response team in a city (p. 75). The book states, “An emergency response unit is to be built that will best serve a city. The goal is to provide the minimum response time to a medical emergency that could occur anywhere in the city.” Using available data and some outside research, engineers were able to construct an algorithm that found optimal locations for the emergency response units. This model is extremely similar to this project's proposed goal: finding the optimal sign location (given landmarks and other signs and cost, etc.). Given that this kind of model has already been done for more complex entities, it should not be a problem to do the same for informational signs. On page 81, the book cites an example where genetic algorithms are used to find optimal antenna array designs. Given all the conditions and possible parameters that go into antenna design, there are over 10^{23} possible designs (considering the data and parameters they used) (p. 85). Unlike any linear methods (or simply guessing), the program can rapidly search through the possible solutions and find the most optimal ones. This indicates that I can prototype new sign layouts very quickly (within a few hours).

Yet another example of the use of genetic algorithms is described in an *Environmental Technology* article by F. Jiang. The scientists who authored this article desire to acquire an, “Understanding [of] pollutant transformation in sewers [because it] is important in controlling odor emission from pressure mains as well as in assessing organic pollutant removal capacity of gravity sewers.” The article gets into detail about how to characterize all the kinetic biological/chemical interactions between bacteria, local pollutants, and oxygen levels. In this instance, scientists were able to use a genetic algorithm they designed to determine the relevant parameters to examine in the pollutant transformation process. This is useful for my project because, like these scientists, there are so many possible parameters to look at for sign distribution (amount of money available for signs, the popularity of certain parts of the trail, sign density and distance to local cultural points-of-interest etc.). The more parameters I consider, the longer the program takes to run. Using the techniques described in this article, I can determine what parameters are most important to sign layout (or to the community) and streamline the development process.

A third scholarly article I acquired discusses the use of genetic algorithms to

automatically compose music. The scientists who wrote this article use models of grammar evolution derived from existing musical melodies (i.e. how note and rest patterns change over time) and then use them to create a new piece of music that sounds like it was written by a human composer (according to them). This article proves that genetic algorithms are capable of optimizing the organization of semantic information. The extensive power of genetic algorithms, as demonstrated by these three articles, proves their usefulness and flexibility in tackling a wide variety of issues.

Genetic Algorithms as they Apply to Otsiningo Park's Trails

The decision to develop a quantitative tool for park development arose from a discussion with Binghamton Youth Bureau Director Ana Shaello Johnson during the first CIC-LI class trip to City Hall on February 1st, 2008. Ana and her colleagues Mike Bowers (of the New York State Department of Transportation) and Gail Domin (Executive Director of the Susquehanna Urban Cultural Park project) had indicated that they were looking for ways to develop functional signage for Binghamton's current and proposed parks and greenways. Ana and I both agreed that while a lengthy qualitative research paper may look nice, the City would benefit much more from a small quantitative (and quantifiable) project with immediate results or usefulness. The Bioengineering program at Binghamton University had taught me several skills that I could apply to such small projects, so I decided to put them to use on improving the area's greenways. I specifically decided to use Otsiningo Park as an example for this project because it is one of the largest parks in Binghamton. I am personally familiar with its trails and cultural events (like the Southern Tier AIDS Program's "Doggone Fun on the Run" in September, 2007 and the annual Spiede Fest).

In an effort to assist me, Mike Bowers forwarded me a 50-page study entitled *Signage and Wayfinding Study for the proposed Shoreline Trail* that was prepared in 2004 by professional architects and engineers for the New York Department of Transportation and Erie and Niagara County, among other beneficiaries (NYS DOT, 2004). Much of the report discussed some issues to consider with physical sign design, but there was very little quantitative analysis on exactly

what constitutes the “best” sign design aside from a financial analysis (which was very thorough.)

The only information available on the physical placement of the signs was presented on page 26:

...Signage interdependence, as described [in this study], places a great deal of importance on programming, or the planning of precisely where signs are placed and what messages they communicate. In order to prevent costly and confusing duplication of information, the programming of signage should follow a “Master Plan” that articulates what destinations are called and what are the best ways to get visitors to those points.

The paper goes on to discuss how the type of sign (directional or informational), exactly what is on the sign and a whole host of other issues are crucially important in ensuring the best layout and distribution of signage. However, in the paper (and the above excerpt) they insinuate that there is one right way to do this (one “Master Plan”). It appears that this Master Plan involves doing a lot of thinking and research without any testing. While studies such as these present a whole host of useful financial and qualitative information, they do not allow city planners to perform quantitative simulations to see what things would look like.

With the genetic algorithm simulation program I created for this project, one can directly simulate what a signage distribution would actually look like (in a relatively short period of time without a lot of manual work). While the model, as it currently stands, is very simplified and does not take into account several of the considerations mentioned in the Wayfinding Study, the results should be enough to demonstrate its potential as a useful tool (with further modification and development). Whereas in the past city planners would have to rely on outsiders to do feasibility studies, an advanced version of this program could allow planners to test their ideas out in-house and in real-time. One could go so far as to implement a 3-dimensional simulation component to have a ground-level view of these sign distributions, but that, while certainly possible, is beyond the current scope of this project.

Model Assumptions, Description and Outlook

The stated goal of this project and model is to design a tool that city planners or a consultant could use to “simulate” potential optimal sign layouts depending on what parameters they think is important. While Otsiningo Park is used as an example for demonstration purposes, the program could conceivably be used for any of Binghamton's parks or greenway trails. The current model will define optimal signage distribution based upon the following factors:

1. Frequency of foot or car traffic on a given area of a trail
2. Cost of signs and available budget
3. Distance between signs and sign density in park

The “traffic” factor will be an example of one way an area of a trail can be “graded.” A portion of the trail with higher traffic would obviously be a better place for a sign than a portion with little or no traffic. In the future models, a user can score trail sections based on their proximity to historical sites or nearby cultural events. The cost of signs/available budget are parameters that can be changed depending on budget constraints for city planners. These parameters control *how many* signs are used in the model. Thus, the program will give planners or consultants the flexibility to optimize the locations of either a small or large number of signs. The program will also be instructed to maximize the distance between signs because it would prove to be redundant and cost-inefficient to have several signs right on top of each other.

As previously indicated, solutions that the program comes up with (possible sign layouts) are graded using something called a fitness function. The fitness function I used is based upon the one used by Randy Haupt in *Practical Genetic Algorithms* when describing the optimal positioning of emergency response teams in a city (Haupt, 80).

$$\text{cost} = \sum_{n=1}^Q t_n \sqrt{(x_n - x_s)^2 + (y_n - y_s)^2}$$

where:

cost = “cost” of putting a sign in a given location (i.e. lower cost is better location)

Q = total number of locations or discretized trail sections to examine

n = the location of the current trail section being examined

t_n = foot traffic level in location n

(x_n, y_n) = coordinates of the center of trail location n

(x_s, y_s) = coordinates of the proposed sign

This fitness function is used to find the theoretical “best” location for a single sign given traffic level patterns. The number of solutions (i.e. sign locations) the program will select will depend on how many signs city planners can afford. For example, if they can only afford one sign, the program will return the best location for that sign on the entire map. If planners can afford 100 signs, the program will find the 100 “best” locations. Engineers and biologists call this type of selection “elitism” as it picks the best solutions to a problem out of a larger pool.

Since most genetic algorithm programs (including this one) are designed to optimize fitness, the fitness function will be the inverse of the cost function described above (1/cost). Thus, the “best” signs will be the ones that minimize the cost of “neglecting” certain people along the trail. In other words, you do not want to position a sign so close to a high traffic area that it neglects people in a low traffic area. The sample run of this model will use a starting population of 100 parents, where each parent is a random coordinate of a “square” on the Otsiningo Park trail. The cost/fitness of these possible sign coordinates will be measured using a form of the equation above and the most “fit” signs (the ones closest to the optimal positions on the map) will be used as the “parents” of the next generation of possible sign coordinates. For those familiar with genetic algorithm programming, the crossover probability and mutation rates were set at 0.6 and 0.004 respectively. Further information can be found in the Appendix.

Initial Results

Using Adobe Fireworks imaging software and *Mathematica 6.0*, a software development platform, I digitized a trail map of Otsiningo Park that I acquired from the Broome County Parks & Recreation website (see Figure 1).



Figure 1. Otsiningo Park Trail guide as presented on GoBroomeCounty.com website

To prepare this map for use in my program, I first had to remove all extraneous information. While the information on the map might be useful later, initially the focus will be entirely on the trail paths themselves. With the trail map streamlined (see Figure 2 below), the map image is prepared for digital processing.

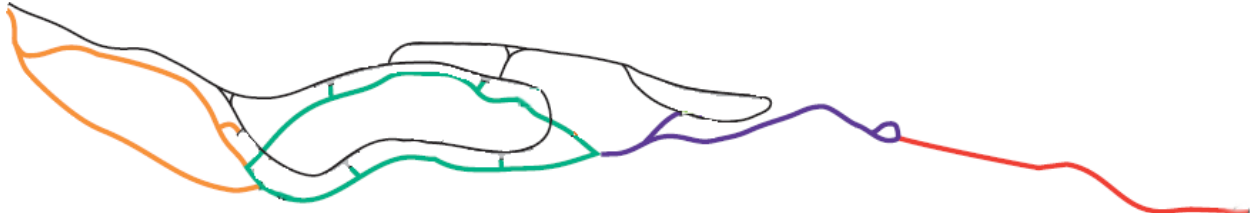


Figure 2. Streamlined version of Otsiningo Park Trail guide.

Once this image is imported into my development software, I instruct the program to subdivide the image into a grid of several squares where a black square is a part of a trail and a white square is not. This helps the computer focus more closely on the trail spaces themselves. In addition, by discretizing the trail map, I can assign values to various areas along the trail. For instance, one area may have more foot traffic than another area (thus it would be a better location for a sign). In addition, I can score each subsection of the trail based on how close it is to a cultural event or point-of-interest. Figure 3 (see below) is a graphical representation of this discretized map.

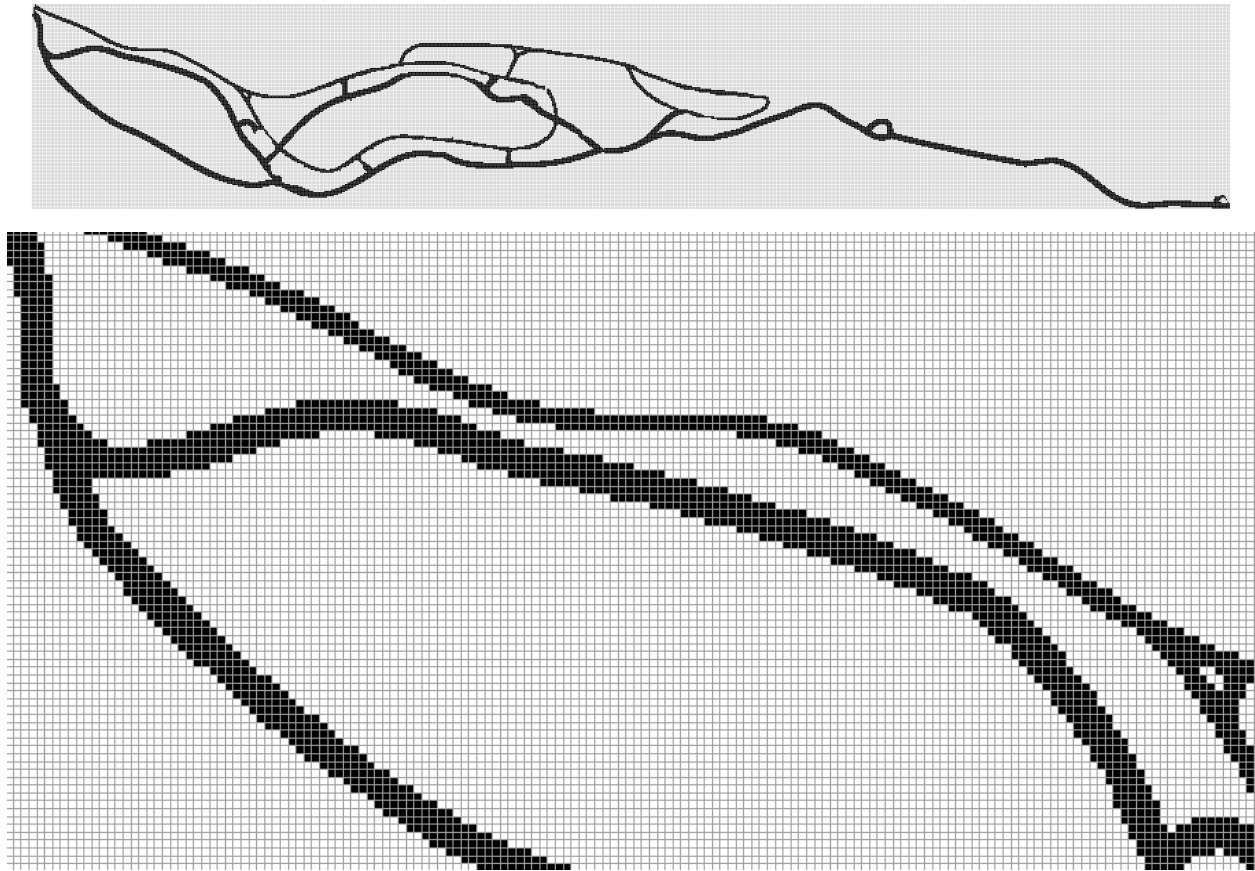


Figure 3. (Top) A miniature version of the discretized Otsiningo park trail map. (Bottom) A close-up view of the same map that allows one to see the individual “pieces” of the trail.

Next, I use an arbitrary function to assign “foot traffic levels” to each part of the map. The relative traffic levels are indicated by the color of the trail section where red areas have the highest traffic and lighter areas have lower traffic. Purple areas are assumed to have zero traffic and are irrelevant at this stage (see Figure 4 below). I use an arbitrary traffic allocation in this simplified example because actual foot traffic pattern data was not available to me at the time of this project. Once this tool is complete, it is simply a matter of gathering the available data and “re-scoring” each section of the trail based on it (i.e. real traffic levels, the direction of the traffic, proximity to events, etc.). Essentially, this program is as powerful and relevant as the data it is fed.

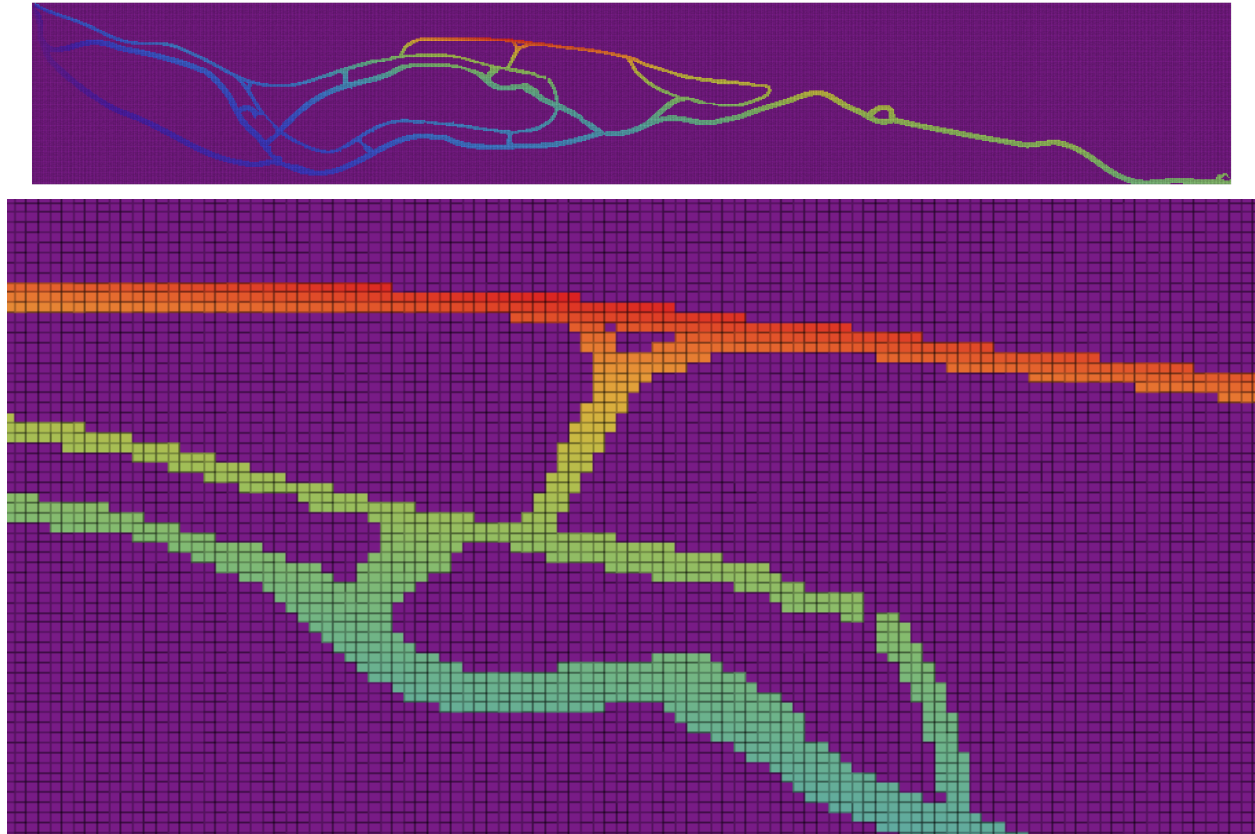


Figure 4. (Top) A miniature version of the discretized Otsiningo park trail map with foot traffic patterns. (Bottom) A close-up view of the same map that allows one to see the individual “pieces” of the trail.

Using the parameters described above, the algorithm was allowed to run for 30 generations. The program was designed in such a way that a user can see the Top 5” coordinates from each generation. If a user wanted to find the optimal position of ten signs, the program can be set to find the “Top 10” instead. Figure 5 shows excerpts of example output. Note that a genetic algorithm does not always progress lineally towards optimization and may have generations where the average fitness is lower than the previous generation.

Generation # 1 (Top 5)	
Fitness = 4.66923×10^{-12}	Coordinates: {95,627}
Fitness = 1.05551×10^{-11}	Coordinates: {100,445}
Fitness = 4.42359×10^{-11}	Coordinates: {96,80}
Fitness = 2.07809×10^{-11}	Coordinates: {111,291}
Fitness = 2.06252×10^{-11}	Coordinates: {99,260}
...	
Generation # 10 (Top 5)	
Fitness = 5.05402×10^{-7}	Coordinates: {97,206}
Fitness = 5.50448×10^{-7}	Coordinates: {103,300}
Fitness = 5.53002×10^{-7}	Coordinates: {97,314}
Fitness = 5.51047×10^{-7}	Coordinates: {61,300}
Fitness = 4.53164×10^{-7}	Coordinates: {61,157}
...	
Generation # 30 (Top 5)	
Fitness = 5.12311×10^{-7}	Coordinates: {17,300}
Fitness = 5.34413×10^{-7}	Coordinates: {106,362}
Fitness = 5.06251×10^{-7}	Coordinates: {93,206}
Fitness = 5.54107×10^{-7}	Coordinates: {93,300}
Fitness = 5.51047×10^{-7}	Coordinates: {61,300}

Figure 5. Generational fitness status of each generation. Because of the nature of the program, the “Top 5” may include duplicates of certain coordinates. If this occurs, one can either choose the next “best” coordinate (e.g. number 6) or simply modify the program to prevent this from happening in the first place.

After the program completes its optimization of the coordinates, it displays a graph indicating the progression of the average fitness of the coordinate populations during each generation. Programmers use this graph as a diagnostic to determine whether their algorithms are really reaching optimal conditions (or something close). A user should be able to see a progression towards a higher average fitness with each generation (see Figure 6). A convergence at a certain average value over several generations may indicate that an algorithm has converged on the most optimal solution it is capable of. The model used for this case study is a work-in-progress and would need to be more thoroughly tested before being used in any official application.

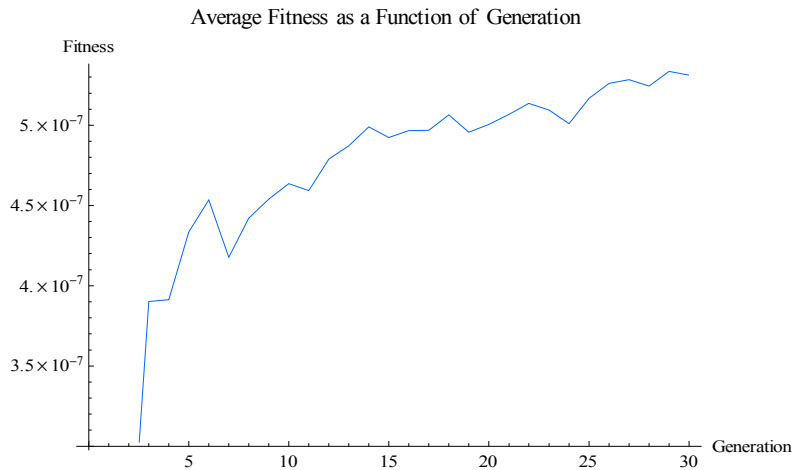


Figure 6. The average fitness of the coordinate population (given the arbitrary traffic distribution shown above) over 30 generations. Note that the fitness generally increases over time and appears to converge at a certain fitness level.

I can use the final “Top 5” list to find where the program thinks the best places are for signs to go given the current traffic distribution. These coordinates are then highlighted in pink on the traffic density map for visual reference (see Figure 7).

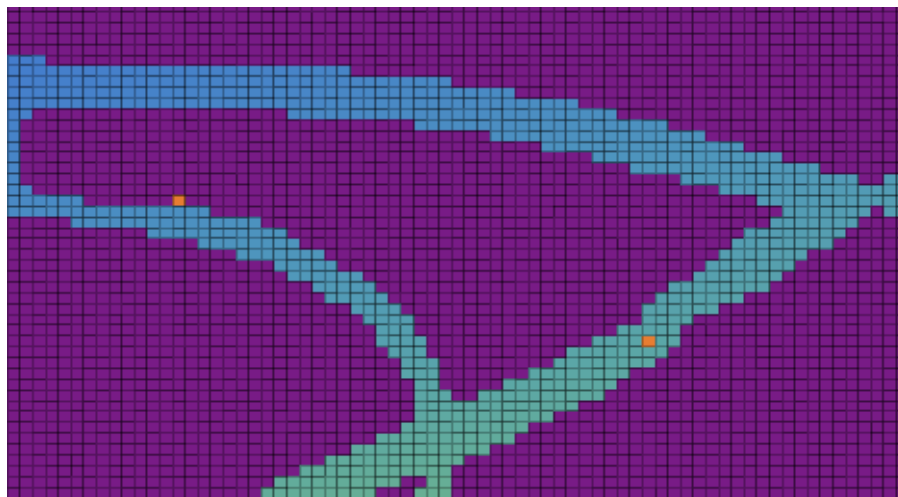


Figure 7. The optimal sign locations are given in pink. Because of the current crudeness of the program, the “sign” locations are not always directly on the trail paths themselves. For the purposes of this demonstration, we will assume that the optimal location is the closest point to the pink “sign dot” that is on a trail.

Because the pixelated output is so small, the program's results are used as a guide for a more eye-friendly display that uses a satellite image of Otsiningo Park (courtesy of Google Earth) and large yellow indicators manually placed over the approximate locations (see Figure 8).



Figure 8. An aerial view of Otsiningo Park. The large yellow diamonds indicate the optimal sign locations based on the program's output. Since the image above was created manually, it is only an approximation.

The results look interesting and appear to correspond to the traffic patterns displayed earlier. Since the signs are spaced adequately apart yet still correspond to traffic patterns, it does not appear that the fitness function needs to explicitly “force” the signs to spread out. In the future, another algorithm could be implemented in this program to maximize the number of signs based on complicated budget constraints and other financial factors. For the purposes of simplicity, I assumed that the City of Binghamton only wanted to place five signs in the park.

The example results cited in this paper should serve to demonstrate the potential usefulness of this as a consulting tool to Binghamton City planners. If real foot or vehicle traffic information were made available, it would not be much trouble to implement the real data and

get useful results. It is recommended that the program undergo further testing with several parameter values (and maps) before being used in official business.

Relevance of Results to Local and Academic Community

This example demonstrates that with a relatively small amount of effort, advanced evolutionary techniques can be used as innovative tools in urban planning situations. With some further development and the addition of real traffic data, this program can easily become a valuable tool for rapid prototyping of development ideas in the Southern Tier (especially when taking into consideration more complex traffic patterns and the proximity of local businesses/fairs/other events etc.).

This project also serves as an academic case study of how relatively abstract engineering techniques can be applied to benefit and interface with local governments. In addition, it is one of the first examples that illustrate how Bioengineers/Complex Systems Engineers (from Binghamton University) can apply their unique skill base in the real world. If the community is able to assimilate this tool into their own research and development, it can serve as an example of the usefulness of genetic algorithms in more “down-to-earth” projects. Much of the current literature on GAs describes their use in technical projects (such as optimization of fiber optic cable layouts) or projects where much of the work is hidden from public view.

As mentioned earlier, the success of this project has several immediate benefits for the Binghamton region. Even a crude planning tool (such as this program, in its current form) might save Binghamton money by allowing city planners to simulate their ideas directly without hiring outside consultants to perform long studies. Binghamton University students that possess skills like my own could even intern with the local government and provide invaluable consultation services at little or no cost.

Possible Expansion & Recommendations

There are several possible areas for the expansion of this project:

- Once real data is inputted into the program, it can become an important tool for further development of the greenways.
- The entire process used to develop this tool (as indicated in this paper) can be streamlined and automated so the program could be used by city officials with less technical skills.
- If the city is ever able to monitor (perhaps using cameras) foot traffic in real time, then signs could be dynamically updated (much like highway traffic signs) so important information gets to the areas where the most people are.
- Since the Binghamton City Police Department have already installed “crime monitoring” cameras in the downtown area, this program could be expanded and modified to allow officials to find the optimal positioning of officers in the City area rather than signs on a greenway trail.

With further work and research, this program, or its derivatives, could become very useful in several aspects of city development and administrative decisions. Successful implementation of any one of these potential uses would not only serve as an interesting academic case study, but also as an indication that Binghamton is serious about finding innovative and creative ways to improve the economy and quality-of-life for its residents.

My most immediate recommendation to any local businessmen, politicians or other community leaders reading this paper is to seek out and invest your time and resources into students with skills like me (from Binghamton University's Bioengineering program). Our entire program revolves around taking such techniques as genetic algorithms and being able to rapidly implement them in ways that can benefit both the private and public sectors. Working with students will almost always be cheaper than hiring outside contractors and will provide greater opportunities for innovative developments.

Appendix A

The following is the Mathematica 6.0 code used to construct the program referenced in this paper. The evolutionary programming techniques implemented in this code were developed and tested in Dr. Walker Land's BE 302 (Adaptive Systems) class at Binghamton University.

Further assistance with interpretation of this code (or a live demonstration) can be arranged by contacting Josh Brandoff at jbrandoff@binghamton.edu

```

(*Import trail Mmp with all extraneous graphics and text manually removed*)
trail = Import["C:\Users\Josh\Documents\Classes\CIC2020\Research\Code\guidegrid.png"]

(*Re-import trail image as a list of color specifications -- turn white space into 0's and everything else into 1's*)
mymap = Import["C:\Users\Josh\Documents\Classes\CIC2020\Research\Code\guidegrid.png", "RGBColorArray"];
For[i = 1, i ≤ Dimensions[mymap][[1]], i++,
  For[j = 1, j ≤ Dimensions[mymap][[2]], j++,
    mymap[i, j] = ToString[mymap[[i, j]]];
    If[mymap[[i, j]] == "RGBColor[1., 1., 1., 1.]", mymap[[i, j]] = 0, mymap[[i, j]] = 1]
  ]];

(*Re-visualize trail map in binary pixelated form*)
ArrayPlot[mymap, Mesh → All, ImageSize → 6000, Frame → False]

(*Use arbitrary function to assign foot traffic values to each portion of the trail*)
For[i = 1, i ≤ Dimensions[mymap][[1]], i++,
  For[j = 1, j ≤ Dimensions[mymap][[2]], j++,
    If[mymap[[i, j]] == 1,
      mymap[[i, j]] = (j / i) * 10 000,
      mymap[[i, j]] = 1]
  ]];

(*Re-visualize trail map with traffic*)
traffic = ListDensityPlot[mymap, ColorFunction → "Rainbow", Mesh → All, InterpolationOrder → 0, Frame → False, AspectRatio → .15, ImageSize → 6000]

(*Create a list of all coordinates on the map which are on trails*)
smartlist = {};
(*Create a list of non-zero coordinates*)
For[i = 1, i ≤ Dimensions[mymap][[1]], i++,
  For[j = 1, j ≤ Dimensions[mymap][[2]], j++,
    If[mymap[[i, j]] > 1, smartlist = Append[smartlist, {i, j}];
  ]];

(*The functions and modules below are part of the actual genetic algorithm program. The code above is used to prepare the map for
assimilation by the rest of the program.*)

(*Flip Function-- Coin-toss function*)
flip[x] := If[Random[] <= x, True, False]

(*XOR T/F function*)
myXor[x, y] := If[x == y, 0, 1];

(*Mutation Algorithm*)
mutateBGA[pmute, allele] := If[flip[pmute], myXor[allele, 1], allele];

(*Roulette Wheel Selection Module*)
selectOne[foldedFitnessList, fitTotal] :=
  Module[{randFitness, elem, index},
    randFitness = RandomReal[] fitTotal;
    elem = Select[foldedFitnessList, # ≥ randFitness & , 1];
    index = Flatten[Position[foldedFitnessList, First[elem]]];
    Return[First[index]];
  ];
```

```

(*Crossover Module*)
crossOver[pcross_, pmutate_, parent1_, parent2_] :=
Module[{child1, child2, crossAt, lchrom},
  (*chromosome length*)
  lchrom = Length[parent1];
  If[flip[pcross],
    (*True:select cross site at random*)
    crossAt = Random[Integer, {1, lchrom - 1}];
    (*construct children*)
    child1 = Join[Take[parent1, crossAt], Drop[parent2, crossAt]];
    child2 = Join[Take[parent2, crossAt], Drop[parent1, crossAt]];
    (*False return parents as children*)
    child1 = parent1;
    child2 = parent2;
  ];
  (*perform mutation*)
  child1 = Map[mutateBGA[pmutate, #] &, child1];
  child2 = Map[mutateBGA[pmutate, #] &, child2];
  Return[{child1, child2}];
];

(*Population Initializer -- customized for otsiningo map size but can be generalized later -- only uses populations of areas on trail*)
initPop[psize_] := Table[smartlist[[RandomInteger[{1, Length[smartlist]}]]], {psize}];

(*Fitness Sorter*)
displayBest[fitnessList_, number2Print_, myPheno_] :=
Module[{i, j, sortedList, newList},
  newList = Table[{fitnessList[[i]], myPheno[[i]]}, {i, Length[fitnessList]}; (*join pheno and fitness into one list to be sorted*)
  sortedList = Sort[newList, Greater];
  For[j = 1, j <= number2Print, j++, Print["Fitness = ", newList[[j, 1]], "      Coordinates: ", newList[[j, 2]]];
  (*display fitness and corresponding pheno with proper units*)
  ];(*end of For i*)
];

```

```

(*Primary BASIC GA Module -- modified to create listplot of average fitness as function of generation*)
bga[pcross_, pmutate_, popInitial_, fitFunction_, numGens_, printNum_] :=
Module[{i, newPop, parent1, parent2, diff, matches, oldPop, reproNum, index, fitList, fitListSum, fitSum, pheno, pIndex1, pIndex2, f,
  children, averageFitness},
  averageFitness = {}; (*initialize fitness container*)
  oldPop = popInitial; (*initialize first population*)
  reproNum = Length[oldPop]/2; (*calculate number of reproductions*)
  f = fitFunction; (*assign the fitness function*)
  For[i = 1, i ≤ numGens, i++, (*perform numGens generations*)
    pheno = oldPop;
    fitList = f/@ pheno; (*determine the fitness of each phenotype*)
    Print[" "]; (*print out the best individuals*)
    Print["Generation #", i, " (Top ", printNum, ")"];
    displayBest[fitList, printNum, pheno];
    fitListSum = FoldList[Plus, First[fitList], Rest[fitList]];
    fitSum = Last[fitListSum]; (*find the total fitness*)
    averageFitness = Append[averageFitness, Mean[fitList]]; (*find and append average fitness*)
    newPop = Flatten[Table[ (*determine the new population*)
      pIndex1 = selectOne[fitListSum, fitSum]; (*select parent indices*)
      pIndex2 = selectOne[fitListSum, fitSum];
      parent1 = oldPop[[pIndex1]]; (*identify parents*)
      parent2 = oldPop[[pIndex2]];
      children = crossOver[pcross, pmutate, parent1, parent2]; (*crossover and mutate*)
      children, {reproNum}], 1 (*add children to list; flatten to first level*)
    ]; (*end of Flatten[Table]*)
    oldPop = newPop; (*new becomes old for next gen*)
  ]; (*end of For i*)
  (*Display Average Fitness ListPlot*)
  Show[ListPlot[averageFitness, PlotJoined -> True, PlotStyle -> Hue[0.6], AxesLabel -> TraditionalForm /@ {Generation, "Fitness"},
    TextStyle -> {FontFamily -> "Times", FontSize -> 14}, ImageSize -> 600, PlotLabel -> "Average Fitness as a Function of Generation"]]
  ];

(*Fitness Function*)
f[x_] := N[1 / (Sum[mymap[[x[[1]], x[[2]]]] * Sqrt[(x[[1]] - smartlist[[j, 1]])^2 + (x[[2]] - smartlist[[j, 2]])^2])]

(*Create initial population of coordinates*)
initialPopulation = initPop[100];

(*bga call with crossover probability of 0.6, a mutation prob of 0.004, an initial population as defined above, the fitness function as defined above,
the number of generations as 30 and the best list of performers set to display on top 5*)
bga[0.6, 0.004, initialPopulation, f, 30, 5]

(*Show map with optimal spots highlighted. Here, the "top 5" coordinates are manually highlighted on the colored map.*)
mymap[[17, 300]] = 100 000;
mymap[[106, 362]] = 100 000;
mymap[[93, 206]] = 100 000;
mymap[[93, 399]] = 100 000;
mymap[[61, 300]] = 100 000;
traffic2 = ListDensityPlot[mymap, ColorFunction -> "Rainbow", Mesh -> All, InterpolationOrder -> 0, Frame -> False, AspectRatio -> .15, ImageSize -> 6000]

```

Works Cited

- de la Puente, Alfonso Ortega, Rafael Sánchez Alfonso, and Manuel Alfonseca Moreno. "Automatic composition of music by means of grammatical evolution." Proceedings of the 2002 conference on APL: array processing languages: lore, problems, and applications (2002): 148-155.
- Bevilacqua, A., Campanini, R., & Lanconelli, N. (2001). A distributed genetic algorithm for parameters optimization to detect microcalcifications in digital mammograms. *Evolutionary Computation in Image Analysis and Signal Processing*, 3, 278-287.
- Freeman, J. A. (1994). *Simulating neural networks: With Mathematica*. New York, NY: Addison-Wesley.
- Haupt, Randy L., and S. E. Haupt. Practical Genetic Algorithms. '2nd ed'. New York: Wiley-IEEE, 2004.
- Heckerling, P. S., Gerber, B. S., Tape, T. G., & Wigton, R. S. (2003). Use of genetic algorithms for neural networks to predict community-acquired pneumonia. *Artificial Intelligence in Medicine*, 30, 71-84.
- Jiang, F, Leung HW, Li SY, and Lin GS. "A new method for determination of parameters in sewer pollutant transformation process model." **Environmental Technology**. 28(2007): 1217-25.
- NYS Department of Transportation. (2004). *Signage and Wayfinding Study for the Proposed Shoreline Trail* [Study]. Buffalo, NY: Wendel Duchscherer Architects & Engineers.
- "Otsiningo Park Trail Guide." Otsiningo Park. 2008. Broome County Department of Parks & Recreation. 8 Mar 2008 <<http://www.gobroomecounty.com/parks/pdfs/OtsiningoTrailGuide.pdf>>.