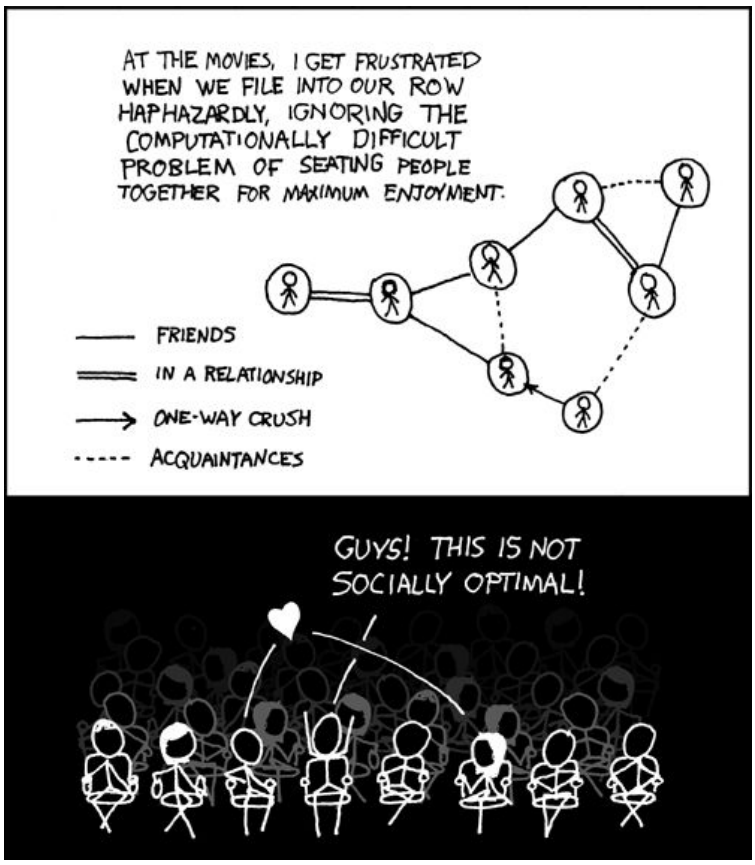
Project 1 Evolutionary Computation

Gerald Hoxha
Bradley Gonthier
Tyler Oliver
Nicholas Van Beek
2/29/2016
CIS412

Part 1

1. Select one problem that is suitable for genetic algorithm (5 points).

The problem that our group chose to solve is the optimal group seating problem. If you have ever been to a group outing, such as a movie/play night, then you likely have experienced this dilemma. You may go out with your significant other and your best friend, plus a group of friends of your best friend who all have their own relationships. How do you seat the group optimally to keep everyone happy?



2. Describe how would you apply genetic algorithm to this problem, i.e.:

How would you code (represent) the solutions? (4 points)

A genetic algorithm is great for this problem because it requires optimization and can easily be tested by a fitness function. A group of 15 people results in 1,307,674,368,000 possible permutations. Genetic algorithms speed up the process by allowing us to narrow down on the more fit seating arrangements and avoid brute force approaches.

The chromosome of our algorithm is directly related to a row of seats. Each person is assigned an integer id. The chromosome is made up of a number of genes equal the number of people we are trying to seat. Each gene is represented by a person's integer id. The location of the gene in the chromosome is where that person will be seated. Example:

Person's integer id: 0 Tyler, 1 Gerald, 2 Bradley, 3 Nick

--0 1 2 3-- This seating order is Tyler, Gerald, Bradley, Nick
--0 2 1 3-- This seating order is Tyler, Bradley, Gerald, Nick
--3 1 0 2-- This seating order is Nick, Gerald, Tyler, Bradley

How would you build the fitness function? (5 points)

In addition to seating position represented by the chromosome, a matrix-esque datastructure denotes each person's relationship to every other person in the group. For example Tyler and Gerald could be friends, while Tyler and Nick are best friends, and Nick and Bradley dislike each other. See the xkcd comic above, the relationship graph represents how it works. Each relationship is represented by a value which is a higher number as the relationship gets stronger. Best friends have a higher desire to sit next to each other than acquaintances. Our relationships are scored like this:

-1: Enemies
0: No relation/preference
1: Acquaintances
2: Friends
3: Best Friends:
4: Lovers

What the fitness function does is takes a chromosome and works from left to right. A variable called “fitness” tallies the total fitness. It takes the left person and figures out the from the matrix what the person’s relationship is to the person on the right. Then it adds the value of the relationship to the fitness tally. So if Tyler and Nick are best friends, and seated next to each other, then add 3 to the fitness. Continue through each gene until the second to last person is reached.

We also created an alternative fitness function that takes into account the seating position of the 2nd person to your right. It weights the person directly right by 2 times the relationship value and the 2nd person to the right by the relationship value.

How would you select the individuals for reproduction and why? (4 points)

Each seating permutation is directly represented by a chromosome. So by comparing two chromosomes with the fitness function, you are comparing which one will render the group more happy as a whole. A higher fitness results in a happier group, so we will be trying to select individuals with the highest fitness.

For example, if we generate a population of chromosomes, we can use tournament selection to randomly select a few chromosomes and compare them to see which one is the most fit and good to go on for crossover based based on the highest fitness value

What is the crossover method and why? (5 points)

We tried a few crossover methods and fully implemented both single point and uniform crossover. Due to the nature of our problem, our crossover method is slightly different than what is traditionally used. The reason why is that duplicate genes in a chromosome are not allowed, because this would result in seating chart where the same person shows up twice in a group. An example of this problem is if we do a single point crossover right down the middle:

--0 1 2 3 | 4 5 6 7-- -----> --0 4 2 7 | 4 5 6 7--
--0 4 2 7 | 6 3 5 1--

If you were to crossover the beginning halves of the chromosomes, then you would end up with duplicate people. For example, person 7 would be present twice in the resulting daughter chromosome. We remedied this by tweaking the traditional crossover methods. What we do is slice one chromosome at a certain point. We take the sliced off section and then find the location of each gene in the slice within the second

chromosome. We pull each of those found genes out of the second chromosome. Then we shift the second chromosome so that all the genes are to one side, and append the sliced part on the end. This prevents duplicates and allows the chromosomes to crossover in a manner that relatively preserves the parents.

How do you implement mutation? (3 points)

For mutation, we randomly swap two genes around inside of a chromosome which randomly changes up the seating position of two people.

What is the termination criterion? (4 points)

It is difficult for the algorithm to know exactly if it has the fittest solution, the only way to know would be to do the problem out by hand to see if the solution is optimal, or have the user of the program intervene when they decide the fittest solution has been found. At any rate, we are always storing a stack of the fittest solutions, and pushing a new one onto the stack when one is found. You can continue running the genetic algorithm for any length of time until you feel that the group has been optimized enough. If a very optimal solution is found, you will see that the fittest solution so far will not be changing as often, this a good termination criterion.

References:

XKCD Comic:
https://xkcd.com/173/

Part 2: Program Usage

Source code with sufficient documentation. Separate report is needed to point out sources you used or any particular set up your program needs to run. The translation from design ideas in Part 1 to implementation in Part 2 should be also documented in this part of the report (item 1).

The genetic algorithm software is built using Javascript which is a web technology. Javascript does not necessarily need to be used in a web browser, but we built the GUI in HTML so in this case it does need to be served to a browser to run. This means in order to run it, you must copy the contents of the project into a directory that is served by a web server such as Node or Apache. Then navigate to index.html in your browser to start using the program.

The Genetic algorithm was created with a helping library called Genetic.JS. Genetic.JS does not implement crossover, mutation or fitness functions. Those were all written by our group. What Genetic.JS does is manage the workflow of creating the population, randomizing when to mutate, choosing when to do a crossover and threading the application to a web worker in order to keep the browser GUI responsive so it is still usable.

Report the results and their analysis under different conditions. Document your simulations describing the used parameters and use charts to illustrate the performance of your algorithm implementation (item 2). The analysis is worth 10 additional points.

We implemented a graph within the GUI that gives you results of the genetic algorithm in action. Within the first screen you will setup your group and all of their relationships. You then select which crossover algorithm, how many iterations, the sample size, crossover rate and mutation probability. Hit run and the genetic algorithm does its work. It outputs a visual of the optimal seating representation found so far, and a chart with minimum, maximum, average, and standard deviation of the fitness for any interval of

generations. We can demo an algorithm and a graph of its finding for a group that needs to be seated.

Single Point vs Uniform Crossover

Uniform crossover had the tendency to create more frequent fluctuations in the average fitness for any generation over the single point crossover. This makes sense, since the genes in a chromosome are much more randomized in their position with uniform

crossover. Single point crossover had the tendency to converge more quickly and find a fitter solution more quickly than uniform crossover. If you take a look at the red line, you can see that the uniform crossover took longer to find an equivalently fit solution to the single point crossover. This is because the uniform crossover better maintains existing chromosomes with great fitnesses, and further uses those to generate more fit offspring.

Low (0.3) vs High (0.7) Crossover Rate:

As you can see from this, the higher crossover rate resulted in convergence on an equally fit solution in less generations. Having a high crossover rate lead to a fit solution more quickly. But higher crossover rate also lead to a lower average fitness. This makes sense, more diversity, so more room for population to have lower fitnesses.

Low (0.3) vs High (0.7) Mutation Rates

Interestingly, the higher mutation rate lead to a less fit final solution after only 200 iterations. While the higher mutation rate achieved it's highest fitness outcome more quickly, it lead to a much less fit solution than the lower mutation rate. Also the higher mutation rate created a much lower average fitness. Having a much lower average

fitness would definitely result in an outcome where it takes longer to find the most fit solution.

Low(250) vs High (500) Sample size

Changing the sample size had a very large effect. The larger sample size result in a much more fit solution. It also had a much higher mean fitness. Even after running the low sample size for 1000 iterations as opposed to the 200 iterations for the high sample size, it still did not reach the same maximum fitness. It was not until nearly 1600 iterations that the low sample size had the same max fitness as the high sample size. That being said, increasing sample size versus adding more iterations may be interchangeable in terms of the amount of real world time and processing power it takes to find a maximum fitness solution.

Fitness functions:

The two fitness functions are slightly different in the way they calculate fitness, and result in very different fitness scores for the same chromosome. Arguable the two person fitness algorithm results in an overall better group seating optimization because you may all talk and interact with the person 2 seats down from you, and the group you would prefer to be lumped in with will be clustered more closely. The graphs for the two algorithms really cannot be compared, but the seating diagram tends to have a more favorable outcome. People who are best friend with someone, yet also friends with someone else tend to be surrounded more by their friends and best friends, and more separated from their enemies, as opposed to the one person fitness function where sometimes your enemy may sit one seat down from you.

Effort Log:
All members contributed equally.

Sources:
Genetic.JS
http://subprotocol.com/system/genetic-js.html