ECE 3790 Lab 4 – Genetic Algorithm

Submitted by: Richard Constantine #7686561

1. *Did the algorithm run as expected? How did you encode a solution?*

Yes, the program executed almost as expected – i.e. it attempted to solve the minimum cost configuration. One caveat was that it did not optimize as well as the greedy or simulated annealing algorithms (for the control matrix – AdjMatCC.txt) from previous labs – even when both selection via tournament AND selection via roulette was used.

The solutions are encoded similar to previous labs. Each component is designated into 2 bins (either bin0 or bin1) via an array (size = number of components) with each element given a value of 0 or 1 depending on which bin the component belongs to. In this lab, a many such arrays (a.k.a generations or solutions) are stored in 2D array forming a "population" of solutions. Then, by evolving the populations, a better solution can be attained. This program will use 100 solutions per population.

2. *What was your minimum score solution. What was the percent improvement from an original or initial average cost "solution".*

Sample outputs (for these stats) are provided in appendix 1.

The minimum score achieved was a cost of 1480 when processing AdjMatCC.txt. This amounts to a ratio/percent improvement of 1.51.

Percent Improvement

Adjacency Matrix	Greedy	SA	GA (Tournament)	GA (Roulette)
AdjMatAsym.txt	190%	185%	131%	143%
AdjMatCC.txt	336%	Infinity	151%	166%
AdjMatRand.txt	128%	143%	155%	126%

3. *How long did the algorithm take to run? If you doubled the size of your problem did the running time scale linearly, quadratically or by some other means?*

Random adjacency matrices were generated using the matrixGenerator.java code (from as lab 1, 2, & 3) and are all provided within the lab package. The were created using 50% sparsity (same test matrices used from lab 2 & 3). Sample outputs of the program executing on the adjacency matrices of size 100 and 200 are provided in Appendix 2 (the matrices used are also provided as attachment).

Average Run-time

Size	Greedy	SA	GA (Tournament)	GA (Roulette)
100 components	1.09 seconds	32.8 seconds	15.5 seconds	9.94 seconds
200 components	2.63 seconds	99.5 seconds	65.2 seconds	82.0 seconds

The run time should scale at least quadratically, however, because the run-time is affected by how the cost changes (stopCount is iterated when the cost/fitness does not change between iterations), this leads to **large** amounts of variation. The largest operations use 2 nested loops (like calculating cost and the mutation function) and therefore should scale by about $O(n^2)$ with the problem size (n).

4. *Vary the component population size. How did this affect the running time?*

Continuing using the previous data and the same matrixGenerator.java (with a sparsity of 50%), the problem size can be varied to observe average run-time (the matrices used are also provided – same used in lab 2). Note, large variance due to how stop count is calculated.

Size	Run-Time (Greedy)	Run-Time (SA)	Run-Time (GA - Tourney)	Run-Time (GA - Roulette)
100 components	1.09 seconds	32.8 seconds	15.5 seconds	9.94 seconds
110 components	1.20 seconds	33.9 seconds	18.7 seconds	11.6 seconds
120 components	1.27 seconds	48.6 seconds	21.9 seconds	17.7 seconds
130 components	1.40 seconds	51.0 seconds	24.8 seconds	19.9 seconds
140 components	1.60 seconds	55.8 seconds	30.0 seconds	27.3 seconds
150 components	1.86 seconds	65.7 seconds	31.8 seconds	31.4 seconds
160 components	2.05 seconds	69.7 seconds	32.6 seconds	33.7 seconds
170 components	2.25 seconds	75.9 seconds	46.3 seconds	45.7 seconds
180 components	2.37 seconds	86.0 seconds	53.6 seconds	52.5 seconds
190 components	2.57 seconds	95.7 seconds	60.1 seconds	64.3 seconds
200 components	2.63 seconds	99.5 seconds	65.2 seconds	82.0 seconds

A plot/graph of the results is provided in Appendix 3.

5. *How might your basic algorithm be improved?*

Some improvements that could be implemented:

- Better s*topCount* response relative to how quickly the solution is changing
    - Better fitness averaging
    - Sample more when change is large and less when change is small
    - More intuitive way of detecting how quickly the solutions are changing
- More efficient method of calculating cost and applying mutations (instead of loops both spanning almost the entire adjacency matrix – instead only calculate necessary connections)
- Visualization aid
- Better selection process
- Better fitness calculation (use more heuristics)

6. *What was the mutation operation you used?*

The mutation operation works by comparing a random number between 0-1 to a fixed probability of .1 (i.e. if rand(0,1) < .1) for every component/bit in every solution of the population. Then, if this condition is satisfied, that particular component is swapped bins (i.e. flip that bit).

7. *Speculate where and why GAs may become important in the future. (Hint: What kind of compute paradigm might make GAs practical? E.g. multicore, cluster.GPU?)*

Since hyper-threaded, multicore processing is becoming the norm, and since quick heuristic algorithms are generally preferred to slow optimal algorithms, the GA excels because it uses large amounts of parallelism to accomplish its goal (i.e. can utilize the multicore/hyperthreaded parallel architectures

more effectively). Therefore, the GA is practical when quickness and parallelism are required – however this involves relatively expensive hardware.

8. *Describe a system, inanimate or living, that evolves, gives some specific examples of steps in the evolution.*

Technology is a good example of how systems can evolve. Most historical technology uses manual labor to accomplish its tasks. Things like stone axes, wells, and horse & buggies, all require a biological entitiy to operate.

Much like the algorithm, these inventions were all innovated upon. Any change that brings positive effect tends to stick around. Stone axes became steel axes, wells became plumbing, and horse & buggies were replaced by automobiles.

However, many such inventions were also left by the way-side because better solutions were presented. Things like asbestos, DDT and certain fertilizers have been abandoned for better, all-around solution.

In this way, evolution (i.e. selection of the fittest) applies to technological changes (and this algorithm) in a similar manner to its references in biology.

Appendix 1

Tournament

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
AdjMatAsym.txt

Old Cost = 4448
Solution:
1110010100111001010100111100101100111100000011000001000111111010000101000011000010100010110110100000

New Cost = 3389
Solution:
0101011110011101010011111001000101110100010101111011000111011101010000010110101100000011000000101100 1

Ratio (ie improvement): 1.3124815579817055

Run-Time: 12670639183 nanoseconds
or 12.671 seconds
Program Ends

****

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
AdjMatCC.txt

Old Cost = 2231
Solution:
1100101000100011001110110001001100100010011110110001001101010010010110110111000101101101111100000100

New Cost = 1480
Solution:
1010110000000111001010000000000101001000000110001111001111000111101111100101111111110010111101111000

Ratio (ie improvement): 1.5074324324324324

Run-Time: 23374822818 nanoseconds
or 23.375 seconds
Program Ends

****

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
AdjMatRand.txt

Old Cost = 5025
Solution:
0011111011000011101110101001110011010001001101111111110110110011111011111100101001100110000110111110

New Cost = 3233
Solution:
1100001100110110000001011001110100110110110011101010111100101001100110110110110010000010101001000110 1

Ratio (ie improvement): 1.5542839467986391

Run-Time: 17803199235 nanoseconds
or 17.803 seconds
Program Ends

Roulette

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
AdjMatCC.txt

Old Cost = 2200
Solution:
011011101001011000011000101000010111000101101101101010110000010111010110011100111111101010110001110 0

New Cost = 1534
Solution:
101011011111111111010010100000101111111100111101111000000000100001011000111000110000011000100000011 1

Ratio (ie improvement): 1.4341590612777053

Run-Time: 14942675125 nanoseconds
or 14.943 seconds
Program Ends

****

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
AdjMatCC.txt

Old Cost = 2497
Solution:
101001011111011100010001000100111011101010010011111110111100111110011011010010000011110011110100010 0

New Cost = 1504
Solution:
010111111011010101111111000011011111110010111111010000000100001100110000110000110110100010100000001 0

Ratio (ie improvement): 1.6602393617021276

Run-Time: 29346528669 nanoseconds
or 29.347 seconds
Program Ends

****

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
AdjMatRand.txt

Old Cost = 4248
Solution:

000010101000111000001111100010111111000000101111101101010110001111001000110100000101110000110000101 0

New Cost = 3361
Solution:
101010011000011101001110010110000111011011010100001101011011100001100000000101011100010011011111101 10

Ratio (ie improvement): 1.2639095507289497

Run-Time: 9384622617 nanoseconds
or 9.385 seconds
Program Ends

Appendix 2

Tournament

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
Q3_1.txt

Old Cost = 6501
Solution:
010001000000010110010010111101011101110101100111011110000111101001101101101011001100100101001011110 01

New Cost = 5743
Solution:
010110100011000000011101100100010011101011001010111010010111101001101100100010110110010011101110001

Ratio (ie improvement): 1.1319867664983458

Run-Time: 15481755773 nanoseconds
or 15.482 seconds
Program Ends

****

What is the size of the matrix?
200
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
Q3_2.txt

Old Cost = 27096
Solution:
110000100100110111100010100011100010111110100001100100010101100000011001000111101100101110011000110100 1
001010100011101100101111101000101110110100111110101100100000101010000100001101000000101011101001000 0

New Cost = 24627
Solution:
101011110111110110110001101001100110100010101100010100111100111011010010000000111000101100100000111110 1
010010011111111110101000101000011000010101011111111011100100000000101101000111001110001001100111000 0

Ratio (ie improvement): 1.1002558167864538

Run-Time: 65178634206 nanoseconds
or 65.179 seconds
Program Ends

Roulette

What is the size of the matrix?
100
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
Q3_1.txt

Old Cost = 6955
Solution:
1100101000011100010011111011101010110010011111010001010001100011010110101010011111101100111101011010

New Cost = 5885
Solution:
1110000110111110000110000011000011010011110110100101001110101010100110011100000001100100010010111111

Ratio (ie improvement): 1.1818181818181819

Run-Time: 9935512486 nanoseconds
or 9.936 seconds
Program Ends

****

What is the size of the matrix?
200
Please enter file name including extension (.txt only) and ensure the txt file is in the same folder as this program.
Q3_2.txt

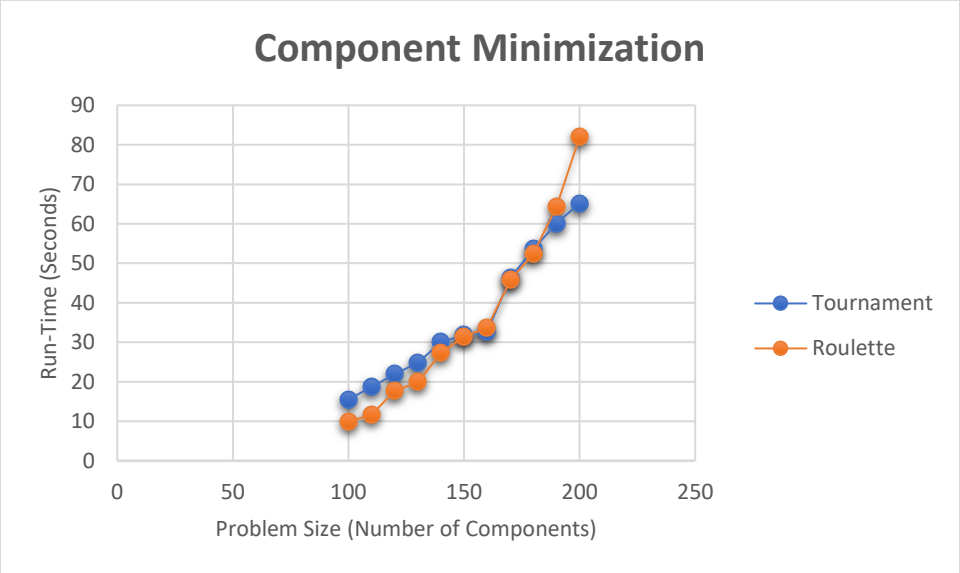
Old Cost = 26650
Solution:
0111011001101100011000101010011110010011000101100010101101110100100010101001100011101001111011111011110001010011101000000101100100000000000000001011111110010100100100001000110100100010011000010111011111

New Cost = 24714
Solution:
0111010000111100110011011000110100110110011111001100011010001000110101010111101101010001101000100011110100110010000101011000100001101111100011000101011001011001001101011010000101010011000101011101111111

Ratio (ie improvement): 1.07833616573602

Run-Time: 82017696203 nanoseconds
or 82.018 seconds
Program Ends

Appendix 3


Component Minimization
Run-Time (Seconds)
90
80
70
60
50
40
30
20
10
0
0
50
100
150
200
250
Problem Size (Number of Components)
Tournament
Roulette