**ECE 3790 Lab 3 Report**

*Submitted by: Richard Constantine*

#7686561

1. *Provide pseudo-code for your algorithm? Did it run as expected? What was the stopping criteria?*

**Input** adjMat, size; [Adjacency Matrix and associated Size]
**Algorithm** simulatedAnnealing
globalStopCount ← 100; [Program will stop using Metropolous criteria – 100 equilibriums in a row with no change]
localStopCount ← 10000; [Number of iterations to establish 1 equilibrium]
penalty ← size/1.5; [Penalty incurred when the cardinality of the bins differs]
bins ← assignBins(adjMat, size); [Assign each component/vertex to a bin – bins are tracked in 1D array]

oldCost ← calculateCost(adjMat, bins, size, penalty); [Calculate cost of initial bin assignment]
print oldCost;
print bins;

newCost ← optimizeCost(adjMat, size, bins, localStopCount, globalStopCount, oldCost); [Optimize via SA]
print newCost;
print bins;

> **Input** adjMat, size;
> **Output** bins;
> **Algorithm** assignBins
> repeat for i from 0 to size - 1
>     bins[i] ← random number between 0 and numBins-1; [In this lab, numBins ← 2]
> end for
>
> **Input** adjMat, size, bins, penalty;
> **Output** cost;
> **Algorithm** calculateCost
> repeat for i from 0 to size-2
>     if bins[i] == 0
>         then bin0count ← bin0count + 1; [Tracks the cardinality of each bin for the penalty]
>     else
>         then bin1count ← bin1count + 1;
>
>     repeat for j from i + 1 to size - 1
>             if bins[i] != bins[j]
>                 then cost ← cost + adjMatrix[i][j]; [Add connections between bins]
>     end for
> end for
>
> if bins[size-1] == 0
>     then bin0count ← bin0count+1;
> else
>     then bin1count ← bin1count+1;
>
> cost ← cost + (|bin0count – bin1count| x penalty); [Add the penalty due to difference in cardinality]
>
> **Input** adjMat, bins, stopCount, oldCost;
> **Output** bestCost;
> **Algorithm** optimizeCost
> globalCount ← 0;
> count ← 0;
> control ← 3000;

```
repeat while globalCount < globalStopCount
        prevCost ← bestCost
        repeat for i from 0 to localStopCount - 1
                rand1 ← random number between 0 and size;
                rand2 ← random number between 0 and size;
                        repeat while rand1 == rand2 || bin[rand1] == bin[rand2]
                                rand1 ← random number between 0 and size;
                                rand2 ← random number between 0 and size;
                        end while

                swapBins(bins, rand1, rand2); [Swap random bins and check the new cost]
                newCost ← calculateCost(adjMat, bins, size, penalty);

                costChange ← newCost – bestCost
                rand3 ← random number between 0 and 1

                if costChange < 0 [Compare the cost, keep any better (or probabilistically better) results]
                        then bestCost ← newCost;
                else if costChange > 0 && rand3 < e^-(costChange/control)
                        then bestCost  ← newCost;
                else
                        then swapBins(bins, rand1, rand2); [Swap bins back]
        end for

        if |bestCost – prevCost| == 0 [If equilibrium has no change, iterate globalCount]
                then globalCount ← globalCount + 1;
        else
                then globalCount ← 0;

        if count mod 2 == 0
                then sample1 ← globalCount;
        else
                then sample2 ← globalCount;

        if |sample2 – sample1| == 0
                then localStopCount ← 50000; [Increase inner loop iterations when solution is changing]
        else
                then localStopCount ← 10000;

        count ← count + 1;
        control ← control *.9;
end while

if (size x penalty) < bestCost [Check if all components in 1 bin yields a better result]
        then print “Penalty may be too low”;
        then bestCost ← (size x penalty);
        then repeat for i from 0 to size - 1
                bins[i] ← 0;
        end for
```

**Input** bins, index1, index2;
**Output** bins;
**Algorithm** swapBins

```
temp ← bins[index1];
bins[index1] ← bins[index2];
bins[index2] ← temp;
```

Yes, the program executed as expected. The stopping criteria of the program is determined by the *globalStopCount*. When the *localCount* (of the inner loop) reaches the *localStopCount*, it compares the previous equilibrium to the currently established equilibrium and iterates *globalCount* when no changes occur. The program then exits when 100 equilibriums have been established (in a row) without any changes – i.e. when *globalCount* reaches *globalStopCount*.

2. *<u>What was your minimum score or solution. What was the percent improvement from and original or initial "solution".</u>*

Sample outputs of the program for each of the provided adjacency matrices can be found in Appendix 1.

The minimum score achieved was a cost of 0 when processing AdjMatCC.txt. This amounts to a ratio/percent improvement of infinity.

| <u>Adjacency Matrix</u> | <u>Percent Improvement (Greedy)</u> | <u>Percent Improvement (SA)</u> |
|---|---|---|
| AdjMatAsym.txt | 190% | 185% |
| AdjMatCC.txt | 336% | Infinity |
| AdjMatRand.txt | 128% | 143% |

3. *<u>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?</u>*

Random adjacency matrices were generated using the matrixGenerator.java code (from as lab 1) and are all provided within the lab package. The were created using 50% sparsity (same test matrices used from lab 2). 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).

| <u>Size</u> | <u>Run-Time (Greedy)</u> | <u>Run-Time (SA)</u> |
|---|---|---|
| 100 components | 1.09 seconds | 32.8 seconds |
| 200 components | 2.63 seconds | 99.5 seconds |

The run time appears to scale almost quadratically (i.e. double problem size, 4 times longer run time). This is because the algorithm uses 2 nested loops in order to calculate cost, therefore, the run-time <u>should</u> scale by about $O(n^2)$ with the problem size (n).

4. *<u>Vary the component population size. How did this affect the running time?</u>*

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 many random influences.

| **Size** | **Run-Time (Greedy)** | **Run-Time (SA)** |
|---|---|---|
| 100 components | 1.09 seconds | 32.8 seconds |
| 110 components | 1.20 seconds | 33.9 seconds |
| 120 components | 1.27 seconds | 48.6 seconds |
| 130 components | 1.40 seconds | 51.0 seconds |
| 140 components | 1.60 seconds | 55.8 seconds |

| | | |
|---|---|---|
| 150 components | 1.86 seconds | 65.7 seconds |
| 160 components | 2.05 seconds | 69.7 seconds |
| 170 components | 2.25 seconds | 75.9 seconds |
| 180 components | 2.37 seconds | 86.0 seconds |
| 190 components | 2.57 seconds | 95.7 seconds |
| 200 components | 2.63 seconds | 99.5 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 *localStopCount* response relative to how quickly the solution is changing
    - 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 (instead of loops both spanning almost the entire adjacency matrix – instead only calculate necessary connections)
- Visualization aid

## Appendix 1

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 = 4238

Bins:
0100001110010001001011000111111010100010101110011111100010110001001111111111011111001010001010
000011

New Cost = 2294

Bins:
1010101010101010111010101010101010101010101010101010101010101010101010111010101010101011101010
101010

Run-Time: 30202138797 nanoseconds
or 30.202 seconds

Ratio (ie improvement): 1.8474280732345247

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 = 1897

Bins:
1010111000101010011001100101100010111110001001001010000110101010101000101101101010011111110011
100001

New Cost = 0

Bins:
1111111111111111111111111111111111111111111111111100000000000000000000000000000000000000000000
000000

Run-Time: 25608439097 nanoseconds
or 25.608 seconds

Ratio (ie improvement): Infinity

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 = 4038

Bins:
0110000011110011001100101110100000101101100111101010010001011000111010001001010111110100100011
000111

New Cost = 2833

Bins:
0011010000001011010110000000100101000100011010111111010010000101010101101111101110110111110100
100110

Run-Time: 37093944203 nanoseconds
or 37.094 seconds

Ratio (ie improvement): 1.4253441581362514

Program Ends

## **Appendix 2**

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 = 7296

Bins:
1011001101101110101110111101011101010110101000110010011010100011101010011101010101100110011101
111110

New Cost = 6093

Bins:
1010110001110101101100100010111111000000011101011001111011001101100101101101011110111011101011
010111

Run-Time: 32825689794 nanoseconds
or 32.826 seconds

Ratio (ie improvement): 1.1974396848842934

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 = 27748

Bins:
0111111110001010001010011101010100111101011011110000101110011010000100101111111110101000011101

0010000111000101001001111011000111111100100111101111111111111000011000101001010101101000010011101011010010 1

New Cost = 24448

Bins:
0111000110010011101011011001110101110100101111001011000010001011001111111111111001111011010001000100000000111111001101101010110101001001110010111100101111000101011111011010000010010011011011101111111000

Run-Time: 93851563131 nanoseconds
or 93.852 seconds

Ratio (ie improvement): 1.1349803664921465

Program Ends

## Appendix 3