ECE 3790 Lab 2 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 componentOptimization
stopCount ← 10000; [Program will stop after 10,000 unsuccessful attempts IN A ROW at finding a better solution]
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, bins, stopCount, oldCost); [Optimize cost via greedy, random swapping method]
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
errorCount ← 0;
repeat while errorCount < stopCount
        rand1 ← random number between 0 and size;
        rand2 ← random number between 0 and size;
```

```
        repeat while rand1 == rand2 || bin[rand1] == bin[rand2] [Ensures 2 unique components in
                                                                different bins]
                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);

        if oldCost < newCost [Compare the cost and keep any better results]
                then oldCost ← newCost;
        else
                then swapBins(bins, rand1, rand2); [Swap bins back]
                then errorCount ← errorCount + 1;
end while

bestCost ← oldCost;

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
                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 ran as expected. The stopping criteria is determined by the stopCount. When the number of unsuccessful attempts in a row (at acquiring a better solution) reaches the stopCount, the program while terminate and print the best cost found so far. For this lab, I have set the stopCount = 10,000. Note: Because we are using a greedy approach, this count influences the accuracy of the output (at the cost of computational time).

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

Sample outputs of the program for each of the provided adjacency matrices (using a StopCount of 10,000) can be found in Appendix 1.

The minimum score achieved was a cost of 715 when processing AdjMatCC.txt.

Adjacency Matrix	Percent Improvement
AdjMatAsym.txt	190%
AdjMatCC.txt	336%
AdjMatRand.txt	128%

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 code the matrixGenerator.java code provided as an attachment (same as lab 1) using 50% sparsity. Sample outputs of the program executing on adjacency matrices of size 100 and 200 are provided in Appendix 2 (the matrices used are also provided as attachment).

Size	Run-Time
100 components	1.09 seconds
200 components	2.63 seconds

The run time scaled almost linearly with the problem size (i.e. when the problem size was doubled, the run-time almost doubled as well). However, because this program uses a random/greedy swapping method along with many other random elements, and the stopCount is relative to the number unsuccessful (random) swaps, the run-times tend to have quite a large variation.

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 run-time (the matrices used are also provided as attachments).

Size	**Run-Time**
100 components	1.09 seconds
110 components	1.20 seconds
120 components	1.27 seconds
130 components	1.40 seconds
140 components	1.60 seconds
150 components	1.86 seconds
160 components	2.05 seconds
170 components	2.25 seconds
180 components	2.37 seconds
190 components	2.57 seconds
200 components	2.63 seconds

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

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

Instead of using a greedy approach (which usually only acquires a good solution), the algorithm could calculate the cost of each different combination of components (i.e. calculate the cost of every situation using the power set of the bins) to obtain the optimal solution. This would increase the accuracy of the answer, but the would require an impractical amount time to execute when the problem size is large.

Appendix 1

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 = 4138

Bins:
011000001001110001000100110000101110110010001111101001010110011111010110111001000000101011101
0010100

New Cost = 2175

Bins:
101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010001
0101000

Run-Time: 1171713894 nanoseconds

Ratio (ie improvement): 1.9025287356321838

Program Ends

****

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 = 2400

Bins:
100110001011101100010110100111110011101001101010000010010010001110001001100000100111110110001
0100000

New Cost = 715

Bins:
100111111111011111111111111011111111111111111111110000000000000000000000000000000000000000000
0000000

Run-Time: 1080463702 nanoseconds

Ratio (ie improvement): 3.3566433566433567

Program Ends

****

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 = 3711

Bins:
011011110010001011000110001001111110111101010100111111100010100000011101110101000000010010000
1011111

New Cost = 2908

Bins:
100100111111100000000010111101101011101110010100100110110110000111011010110010110000011100010010011001

Run-Time: 1091843693 nanoseconds

Ratio (ie improvement): 1.2761348005502062

Program Ends

Appendix 2

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 = 6211

Bins:
110011010010111111101010000110000111100010011110110100011111011011100010100101010001000000101100111001110

New Cost = 5397

Bins:
001010110100110010101100101011110110101110111111011100111011000100010001000011001000001100010111010101011

Run-Time: 1092333977 nanoseconds

Ratio (ie improvement): 1.1508245321474893

Program Ends

****

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 = 27004

Bins:
100011000011001011111011001100100001000000101101110101111010100101001100011010001010101101011010110111011010000101111010001100000101101001010000111110010000011011100110110001001100101101010011010100000010010

New Cost = 24266

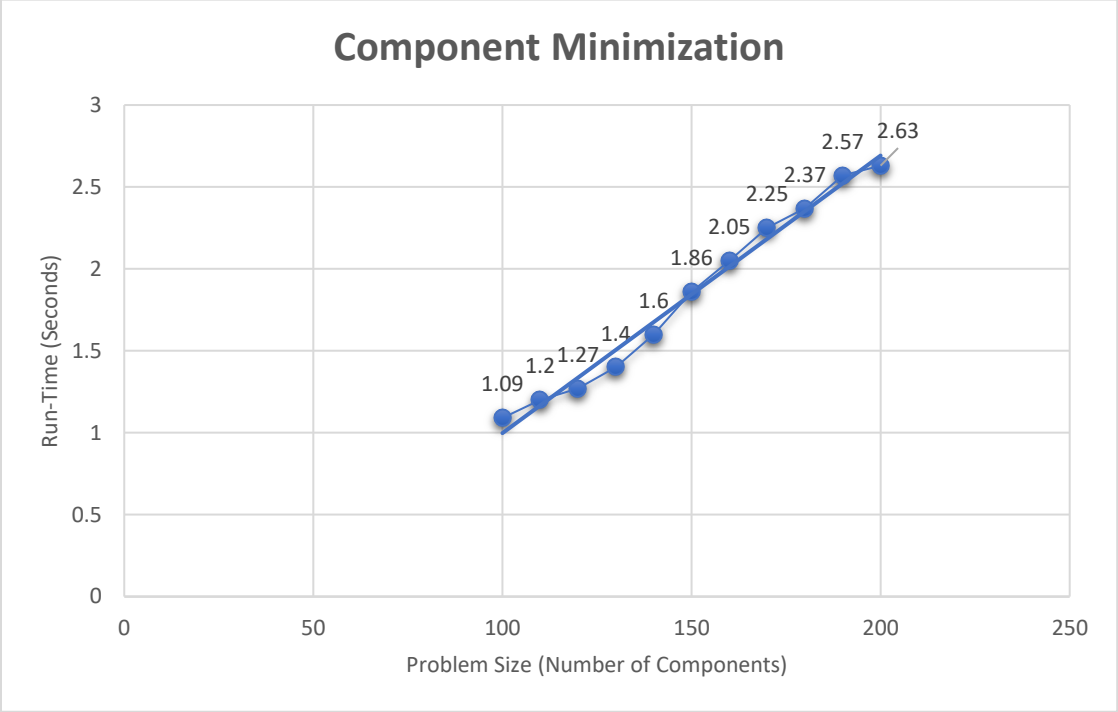
Bins:
001001110100011011001000110000001100001010101100000111111100100010100101011100011000110111011010100111111111101001101011111000101010110010001000001100001100011111011000100010100011001010010011111001100100000

Run-Time: 2632394778 nanoseconds

Ratio (ie improvement): 1.1128327701310476

Program Ends

Appendix 3


Component Minimization
Run-Time (Seconds)
Problem Size (Number of Components)
0
0.5
1
1.5
2
2.5
3
0
50
100
150
200
250
1.09
1.2
1.27
1.4
1.6
1.86
2.05
2.25
2.37
2.57
2.63