University of Waterloo
Faculty of Engineering

Rank Preserving Clustering Algorithms for Paths in Social Graphs

LinkedIn Corporation
Mountain View, CA 94043

Prepared by
Ziyad Mir
ID 20333385
2B Department of Systems Design Engineering
May 7, 2012

400 Joseph Street
Port Elgin, Ontario
N0H 2C2

May 7, 2012

Paul Fieguth, Chair
Department of Systems Design Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Professor Fieguth,

I have prepared this report, "Rank Preserving Clustering Algorithms for Paths in Social Graphs" as my 2B Work Report for the Distributed Cloud/Social Graph team at LinkedIn. This report is the third of four that I must submit as part of my degree requirements, and it has not received any previous academic credit. This report was entirely written by me and has not received any previous academic credit at this or any other institution.

LinkedIn is a business-related social networking site, focused on providing services to the working professionals of the world. Within LinkedIn, the Distributed Cloud/Social Graph team, of which I was a member, is responsible for the social graph infrastructure, constituting the backbone of the site.

This report examines the performance of various algorithms which cluster paths in social graphs, in regards to both worst-case, asymptotic memory and time complexity. It aims to determine the situations in which each algorithm performs well, and seek out the appropriate algorithm for a path-finding task I encountered during the work term.

I would like to thank Chris Conrad, Engineering Manager, and Rui Wang, Staff Software Engineer, for their guidance throughout my internship. I hereby confirm that I have received no further help, other than what is mentioned above, in writing this report.

Sincerely,

Ziyad Mir
ID 20333385

Abstract

The purpose of this report is to analyze the running time and memory usage of different algorithms for path clustering in social graphs. Ideally, a single algorithm should be deemed optimal for the path-finding task at hand, and, as a result, should be recommended for use.

This report begins with an explanation of social graphs - models which represent the interactions and behaviours of individuals, on large-scale, distributed web applications. A description of paths, and path-ranking in social graphs is given, and the problem at hand, relating to clustering ranked paths in social graphs, is introduced. Various algorithms are proposed to solve the problem and each is analyzed, in regards to worst-case, asymptotic running time and memory complexity. Finally, a single algorithm is selected as the most performant, and a recommendation is proposed.

Table of Contents

List of Tables

List of Figures

1.0 Introduction

This report documents a study into various path clustering algorithms, which preserve rankings, for paths in social graphs. It examines a real-world problem faced while developing a distributed path-finding application, and the series of steps taken to determine an optimal solution.

1.1 Background

LinkedIn is an American, professional networking website, with over 150,000,000 users. Within LinkedIn, the Search Network and Analytics (SNA) division is responsible for the infrastructure, search functionality and data-driven products on the site. Inside of SNA, the Distributed Cloud/Social Graph team is responsible for the core graph infrastructure that powers the site. All queries to social graph related features are handled by layers implemented by the Distributed Cloud/Social Graph team.

1.2 Graphs

In computer science and mathematics, a graph is an abstract model, representing sets of objects and connections. The objects, called vertices, are connected by edges.

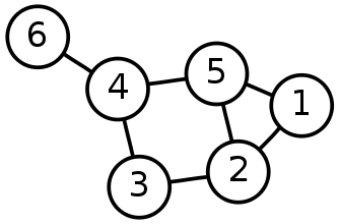

Figure 1: A graph on six vertices and seven edges

In a graph, a path indicates a sense of connectedness, in that if there exists a path betwwen node A and node B, they are considered connected. The following image outlines the concept of a path.

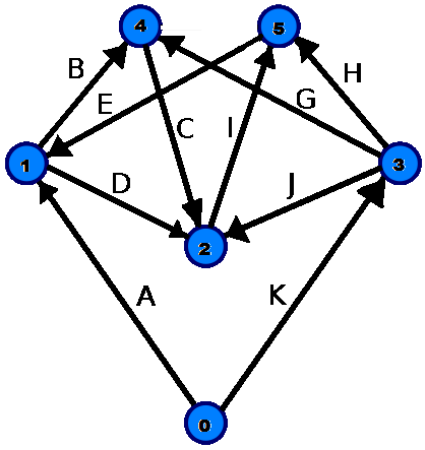

Figure 2: Paths in a graph

In this graph, node 1 is connected to node 4 (there is a path from node 1 to node 4), however, node 5 is not connected to node 0 (there is no path from node 5 to node 0).

Edges in a graph can hold different attributes, including data representing edge scores. A path can combine the edge scores of the edges along the path of interest to determine a path score. This path score can act as a proxy of the strength of a certain value. The exact meaning of the path score is irrelevant for the purpose of this analysis, however, the concept of path's having scores is crucial.

1.3 Motivation and Significance

Social media and networking sites have become overwhelmingly popular in the last decade. Major sites such as Facebook, Google, Twitter and LinkedIn have developed massive plat-

forms on which millions of people interact, on a day-to-day basis. Their social graphs are large and dense. In order to leverage the vast information stored within these graphs, many companies have turned to social graph application development.

One area of interest for companies with large social graphs, is to develop applications based on the concept of paths. At LinkedIn, a path-finding application was developed on top of the underlying social graph. This application aimed at determining paths between nodes, ranking them by path score, and clustering them by path type. Effective path clustering algorithms were needed, in order to allow path-finding on massive social graphs to be feasible.

1.4 Adjacency List Representation of Graphs

To represent a graph, an adjacency list can be used. The list represents the set of nodes that are adjacent to, or reachable, from a specific node.

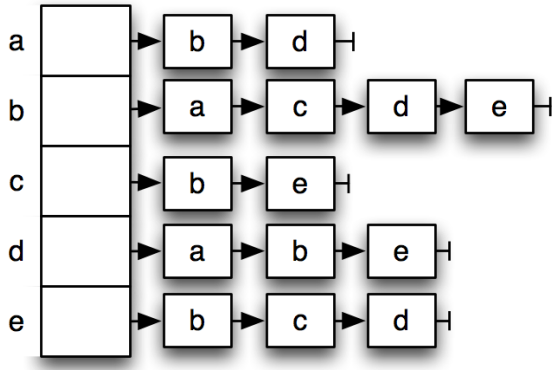

Figure 3: An adjacency list representation of a graph

In this adjacency list, nodes b and d are adjacent to node a, however, nodes e and c are not.

1.5 Path Rankings

For the context of this report, paths will be associated with a single integer, called their path score. Paths will be ranked in non-decreasing ordering, by path score. The path score acts as proxy of the strength of that path, and thus, stronger paths have higher path scores.

1.6 Path Types

For the context of this report, paths will be limited to three specific types, 1-degree, 2-degree, and 3-degree. Assume the starting node of path, P, is A and the ending node is B. Then, a 1-degree path has one edge, E_1 connecting A to B, a 2-degree path has two edges, E_1 and E_2 connecting A to B and a 3-degree path has three edges, E_1, E_2 and E_3 connecting A to B. Auxiliary nodes (nodes along the path from A to B) bear negligible importance in this analysis).

1.7 Cluster Types

For the context of this report, there will be two clustering types considered.

The first, called Path Type Clustering (PTC), groups the set of path results into separate groups, one for each degree a path can have. Thus, this clustering style will yield three clusters of results, one corresponding to a 1-degree path, one corresponding to a 2-degree path, and one corresponding to a 3-degree path. All paths within a given cluster must be arranged in non-decreasing order, according to path score.

The second, called Path Compression Clustering (PCC), groups adjacent sets of path results into groups, based on their path types. All adjacent paths of the same type are

compressed into a cluster containing the aggregate, adjacent paths of their type. All paths must be arranged in non-decreasing order, according to path score.

1.8 Problem Description

Given a set of n paths, P_1, ..., P_n, determine algorithms which allow for effective (in regards to worst-case, asymptotic running time and memory complexity) Path Compression Clustering and Path Type Clustering.

1.9 Approach and Evaluation

In order to determine the performance of various techniques, algorithms will be judged based on their worst-case, asymptotic running time and memory complexity. In the case of shared worst-case running time or memory complexity, algorithms will be judged based on perceived constant factor overhead, based on implementation details. Elementary order analysis will be conducted, to characterize the behaviour of the algorithms, as their input size, n, tends to infinity. The input size, n, will refer to the total number of paths to be clustered.

2.0 Design

Two algorithms are proposed, Path Clustering A (PCA), and Path Clustering B (PCB).

2.1 Path Clustering A Design

A description of PCA being applied to a list of n paths, ordered by path score is as follows.

First, three linked list of pairs of lists of paths and their cluster numbers, LLP_1, LLP_2 and LLP_3, should be created, one for each degree of path. Next, a counter variable, $clusterNum$ should be initialized to zero, indicating the cluster number. A second variable, $offset$, should be created, and should take the value of 0 if the desired clustering type is PCT, and 1 otherwise. Then, while there are paths, in the set of paths, that have not been iterated over, the following procedure will be repeated. While the path type, T_i, of adjacent paths are the same (T_i is equal to T_{i+1}), those paths should be added to a list of paths, L_P. When the path type T_i of adjacent paths differ (T_i is not equal to T_{i+1}), L_P and $clusterNum$ should be inserted as a pair, to the end of LLP_i. Subsequently, $clusterNum$ should be incremented by $offset$, and L_P should be cleared.

LLP_1, LLP_2 and LLP_3, should then be merged, in a stable manner, into an auxiliary linked list, LLP_F, ordered by minimum $clusterNum$.

If $offset$ is 0, all $clusterNum$ values will be 0, and as a result, due to the stability of the merging routing, LLP_F, will contain three separate clusterings of lists of paths. Otherwise, LLP_F will contain at most n clusterings of lists of paths.

2.2 Path Clustering A Implementation

See Appendix B for Path Clustering A implementation.

2.3 Path Clustering B

A description of PCB being applied to a list of n paths, ordered by path score is as follows.

Let $clusterType$ be the desired clustering.

A linked list, LLP_F, of lists of paths should be created. If $clusterType$ is PCT, three linked lists of paths, LLP_1, LLP_2 and LLP_3, should be created, one for each degree of path. Then, while there are paths, in the set of paths, that have not been iterated over, the following procedure will be repeated. While the path type, T_i, of adjacent paths are the same (T_i is equal to T_{i+1}), those paths should be added to a list of paths L_P. When the path type T_i of adjacent paths differ (T_i is not equal to T_{i+1}), if $clusterType$ is PCC, L_P should be inserted into LLP_F, otherwise, L_P should be added to the back of LLP_{T_i}.

If $clusterType$ is PCT, LLP_1, LLP_2 and LLP_3, should then be merged, sequentially into LLP_F.

2.4 Path Clustering B Implementation

See Appendix C for Path Clustering B implementation.

3.0 Analysis

This section contains an analysis of the worst-case, asymptotic run time and memory complexity of both algorithms.

3.1 Path Clustering A Running Time

In the worst-case, each of the n paths in the input list of paths will needed to be iterated through, and thus, PCA runs in $O(n)$ time.

3.2 Path Clustering B Running Time

In the worst-case, each of the n paths in the input list of paths will needed to be iterated through, and thus, PCB runs in $O(n)$ time.

3.3 Path Clustering A Memory Usage

In the worst-case, each of the n paths in the input list of paths will be added to the same linked list, and thus, PCA uses $O(n)$ extra memory.

3.4 Path Clustering B Memory Usage

In the worst-case, each of the n paths in the input list of paths will be added to the same linked list, and thus, PCB uses $O(n)$ extra memory.

3.5 PCA vs. PCB

Since PCA and PCB share the same worst-case, asymptotic running time and memory complexity, an additional analysis must be conducted, to determine which algorithm is preferred. The basis for this analysis will be perceived constant factor overhead (in regards to both memory usage and running time), based on the specifics of implementation for each algorithm.

PCA will use three nearly-equally sized linked lists, in most cases. Furthermore, a three-way merge will be required, in most cases. This merge routine will require simultaneous comparison of the minimum $clusterNum$ at the heads of each list, for every iteration. This repeated computation results in a running time, growing linearly in the number of paths.

On the other hand, PCB will only require three additional linked lists approximately half of the time. For each call to PCB requesting a Path Compression Clustering (PCC), only the minimum auxilliary memory will be used (to store the elements in the current cluster). After the current list is filled, it will immediately be inserted into the final data structure, LLP_F. In the case of a call to requesting a Path Compression Clustering (PCC), PCB will outperform PCA. In the case of a call to PCB requesting a Path Type Compression, a three-way merge will be required. However, in constrast to the merge that PCA requires, this merge will not require any comparisons. Rather, the three linked lists LLP_1, LLP_2 and LLP_3 can be concatenated sequentially. With modern programming languages supporting efficient list comprehension, this operation can be computed in $O(1)$ time, with a fixed number of pointer shifts.

Dominating both types of clustering calls (PCC and PTC), PCB is clearly the more efficient algorithm.

3.6 PCA vs. PCB Testing

See Appendix D for the testing data, outlining an empirical comparison of PCA and PCB. The results are summarized in this section.

Each algorithm was tested 100 times, against an input set of 100 paths. The average memory usage, in bytes, and running time, in seconds, is outlined below.

Table 1: PCA vs. PCB Empirical Results

	PCA	**PCB**
Mean Memory Usage (bytes)	9999.44	5485.90
Mean Running Time (seconds)	207.10	169.37

It is evident that PCB has lower mean running time and memory usage than PCA.

4.0 Conclusions

Upon first glance, PCA might have appeared the superior algorithm, as its elegant design allows both clustering types to be handled by the same routine. However, the aggregation of both types of clustering into one algorithm yields a compromise for both cases. Though the design of PCB may seem less elegant, catering to the needs of each clustering call allows optimizations to be made which, ultimately, yield a more effective algorithm.

Based on the analysis conducted within the body of this report, it is clear that the preferred algorithm is Path Clustering B.

5.0 Recommendations

Based on the direct conclusions of this report, it is recommended that Path Clustering B be used as the algorithm to cluster ranked paths.

Appendix A - Glossary

1. **PTC** stands for Path Type Clustering, a grouping of the set of path results into three separate groups, one for each degree a path can have. This clustering style will yield three clusters of results, one corresponding to a 1-degree path, one corresponding to a 2-degree path, and one corresponding to a 3-degree path.

2. **PCC** stands for Path Compression Clustering, a grouping of adjacent sets of path results into groups, based on their path types. All adjacent paths of the same type and compressed into a cluster containing the aggregate, adjacent paths of their type.

3. **PCA** stands for Path Clustering A, the first of two algorithms proposed.

4. **PCB** stands for Path Clustering B, the second of two algorithms proposed.

Appendix B - Path Clustering A Implementation

```
List<List<Path>> getClusteredPaths(List<Path> paths, String clusterType)
{
  List<List<Path>> LLP_F = new ArrayList<List<Path>>();
  List<Pair<Integer, List<Path>> LLP_1 = new ArrayList<Pair<Integer, List<Path>>();
  List<Pair<Integer, List<Path>> LLP_2 = new ArrayList<Pair<Integer, List<Path>>();
  List<Pair<Integer, List<Path>> LLP_3 = new ArrayList<Pair<Integer, List<Path>>();
  List<Path> LP = new ArrayList<Path>();

  Integer offset = 0;
  if(clusterType.equals("PathCompressionClustering") )
      offset = 1;

  Integer prevSize = 0, curSize = 0, clusterNum = 0;
  boolean first = true;

  for(Path path : paths)
  {
    if(first)
    {
      prevSize = path.getRoutingEdges().size();
      first = false;
    }
    curSize = path.getRoutingEdges().size();
```

```
    if(prevSize.equals(curSize))
    {
      LP.add(path);
    }
    else
    {
      if(curSize.equals(1))
        LLP_1.add(new Pair<Integer, List<Paths>(clusterNum, LP));
      else if(curSize.equals(2))
        LLP_2.add(new Pair<Integer, List<Paths>(clusterNum, LP));
      else
        LLP_3.add(new Pair<Integer, List<Paths>(clusterNum, LP));
    }

    LP.clear();
    prevSize = curSize;
  }

  mergeByMinClusterNum(LLP_1, LLP_2, LLP_3, LLP_F);

  return LLP_F;
}
```

Appendix C - Path Clustering B Implementation

```
List<List<Path>> getClusteredPaths(List<Path> paths, String clusterType)
{
  List<List<Path>> LLP_F = new ArrayList<List<Path>>();

  if(clusterType.equals("PathTypeClustering") )
  {
      List<Path> LLP_1 = new ArrayList<Path>();
      List<Path> LLP_2 = new ArrayList<Path>();
      List<Path> LLP_3 = new ArrayList<Path>();
  }

  List<Path> LP = new ArrayList<Path>();

  Integer prevSize = 0, curSize = 0;
  boolean first = true;

  for(Path path : paths)
  {
    if(first)
    {
      prevSize = path.getRoutingEdges().size();
      first = false;
    }
    curSize = path.getRoutingEdges().size();
```

```
  if(prevSize.equals(curSize))
  {
    LP.add(path);
  }
  else
  {
    if(clusterType.equals("PathCompressionClustering")
    {
        LLP_F.add(LP);
    }
    else
    {
        if(curSize.equals(1)) LLP_1.add(LP);
        else if(curSize.equals(2)) LLP_2.add(LP);
        else LLP_3.add(LP);
    }
    LP.clear();
    prevSize = curSize;
  }
}

if(clusterType.equals("PathCompressionClustering")
{
    merge(LLP_1, LLP_2, LLP_3, LLP_F);
}
```

```
  return LLP_F;
}
```

Appendix D - PCA vs. PCB Testing Results

The log file containing the test results exceeds 60,000 lines. For brevity, only the first 250 lines of the log file are attached.

The complete file is available for download from http://www.eng.uwaterloo.ca/ zmmir/results.log.

```
Test Number 1
Path Number 1
PCA Memory Usage (bytes): 10073
PCA Running Time (seconds): 184
PCB Memory Usage (bytes): 5548
PCB Running Time (seconds): 234

Path Number 2
PCA Memory Usage (bytes): 10088
PCA Running Time (seconds): 142
PCB Memory Usage (bytes): 5451
PCB Running Time (seconds): 227

Path Number 3
PCA Memory Usage (bytes): 9952
PCA Running Time (seconds): 218
PCB Memory Usage (bytes): 5482
PCB Running Time (seconds): 198
```

```
Path Number 4
PCA Memory Usage (bytes): 9971
PCA Running Time (seconds): 210
PCB Memory Usage (bytes): 5584
PCB Running Time (seconds): 259

Path Number 5
PCA Memory Usage (bytes): 10029
PCA Running Time (seconds): 254
PCB Memory Usage (bytes): 5410
PCB Running Time (seconds): 193

Path Number 6
PCA Memory Usage (bytes): 9896
PCA Running Time (seconds): 152
PCB Memory Usage (bytes): 5409
PCB Running Time (seconds): 235

Path Number 7
PCA Memory Usage (bytes): 9926
PCA Running Time (seconds): 186
PCB Memory Usage (bytes): 5407
PCB Running Time (seconds): 86

Path Number 8
PCA Memory Usage (bytes): 10107
```

```
PCA Running Time (seconds): 146
PCB Memory Usage (bytes): 5490
PCB Running Time (seconds): 243

Path Number 9
PCA Memory Usage (bytes): 10024
PCA Running Time (seconds): 163
PCB Memory Usage (bytes): 5516
PCB Running Time (seconds): 175

Path Number 10
PCA Memory Usage (bytes): 9998
PCA Running Time (seconds): 308
PCB Memory Usage (bytes): 5442
PCB Running Time (seconds): 228

Path Number 11
PCA Memory Usage (bytes): 10006
PCA Running Time (seconds): 265
PCB Memory Usage (bytes): 5465
PCB Running Time (seconds): 254

Path Number 12
PCA Memory Usage (bytes): 9953
PCA Running Time (seconds): 175
PCB Memory Usage (bytes): 5553
```

```
PCB Running Time (seconds): 260

Path Number 13
PCA Memory Usage (bytes): 9907
PCA Running Time (seconds): 303
PCB Memory Usage (bytes): 5492
PCB Running Time (seconds): 81

Path Number 14
PCA Memory Usage (bytes): 9934
PCA Running Time (seconds): 242
PCB Memory Usage (bytes): 5571
PCB Running Time (seconds): 137

Path Number 15
PCA Memory Usage (bytes): 9906
PCA Running Time (seconds): 104
PCB Memory Usage (bytes): 5478
PCB Running Time (seconds): 76

Path Number 16
PCA Memory Usage (bytes): 9944
PCA Running Time (seconds): 308
PCB Memory Usage (bytes): 5573
PCB Running Time (seconds): 245
```

```
Path Number 17
PCA Memory Usage (bytes): 9950
PCA Running Time (seconds): 215
PCB Memory Usage (bytes): 5460
PCB Running Time (seconds): 226

Path Number 18
PCA Memory Usage (bytes): 10003
PCA Running Time (seconds): 243
PCB Memory Usage (bytes): 5494
PCB Running Time (seconds): 71

Path Number 19
PCA Memory Usage (bytes): 9986
PCA Running Time (seconds): 300
PCB Memory Usage (bytes): 5579
PCB Running Time (seconds): 217

Path Number 20
PCA Memory Usage (bytes): 9954
PCA Running Time (seconds): 258
PCB Memory Usage (bytes): 5517
PCB Running Time (seconds): 139

Path Number 21
PCA Memory Usage (bytes): 10040
```

```
PCA Running Time (seconds): 135
PCB Memory Usage (bytes): 5474
PCB Running Time (seconds): 251

Path Number 22
PCA Memory Usage (bytes): 10071
PCA Running Time (seconds): 170
PCB Memory Usage (bytes): 5429
PCB Running Time (seconds): 254

Path Number 23
PCA Memory Usage (bytes): 9968
PCA Running Time (seconds): 247
PCB Memory Usage (bytes): 5585
PCB Running Time (seconds): 189

Path Number 24
PCA Memory Usage (bytes): 10034
PCA Running Time (seconds): 284
PCB Memory Usage (bytes): 5474
PCB Running Time (seconds): 261

Path Number 25
PCA Memory Usage (bytes): 9978
PCA Running Time (seconds): 274
PCB Memory Usage (bytes): 5526
```

```
PCB Running Time (seconds): 258

Path Number 26
PCA Memory Usage (bytes): 9996
PCA Running Time (seconds): 146
PCB Memory Usage (bytes): 5584
PCB Running Time (seconds): 260

Path Number 27
PCA Memory Usage (bytes): 9924
PCA Running Time (seconds): 289
PCB Memory Usage (bytes): 5517
PCB Running Time (seconds): 155

Path Number 28
PCA Memory Usage (bytes): 10026
PCA Running Time (seconds): 160
PCB Memory Usage (bytes): 5548
PCB Running Time (seconds): 129

Path Number 29
PCA Memory Usage (bytes): 9988
PCA Running Time (seconds): 148
PCB Memory Usage (bytes): 5420
PCB Running Time (seconds): 122
```

```
Path Number 30
PCA Memory Usage (bytes): 10012
PCA Running Time (seconds): 189
PCB Memory Usage (bytes): 5416
PCB Running Time (seconds): 257

Path Number 31
PCA Memory Usage (bytes): 9915
PCA Running Time (seconds): 127
PCB Memory Usage (bytes): 5486
PCB Running Time (seconds): 226

Path Number 32
PCA Memory Usage (bytes): 10104
PCA Running Time (seconds): 300
PCB Memory Usage (bytes): 5526
PCB Running Time (seconds): 145

Path Number 33
PCA Memory Usage (bytes): 10054
PCA Running Time (seconds): 179
PCB Memory Usage (bytes): 5443
PCB Running Time (seconds): 112

Path Number 34
PCA Memory Usage (bytes): 10018
```

```
PCA Running Time (seconds): 152
PCB Memory Usage (bytes): 5412
PCB Running Time (seconds): 220

Path Number 35
PCA Memory Usage (bytes): 9919
PCA Running Time (seconds): 270
PCB Memory Usage (bytes): 5415
PCB Running Time (seconds): 223

Path Number 36
PCA Memory Usage (bytes): 9909
PCA Running Time (seconds): 304
PCB Memory Usage (bytes): 5391
PCB Running Time (seconds): 175

Path Number 37
PCA Memory Usage (bytes): 9930
PCA Running Time (seconds): 151
PCB Memory Usage (bytes): 5551
PCB Running Time (seconds): 220

Path Number 38
PCA Memory Usage (bytes): 10033
PCA Running Time (seconds): 307
PCB Memory Usage (bytes): 5517
```

```
PCB Running Time (seconds): 226

Path Number 39
PCA Memory Usage (bytes): 9913
PCA Running Time (seconds): 128
PCB Memory Usage (bytes): 5491
PCB Running Time (seconds): 79

Path Number 40
PCA Memory Usage (bytes): 9908
PCA Running Time (seconds): 143
PCB Memory Usage (bytes): 5479
PCB Running Time (seconds): 239

Path Number 41
PCA Memory Usage (bytes): 10016
PCA Running Time (seconds): 262
PCB Memory Usage (bytes): 5391
PCB Running Time (seconds): 96

Path Number 42
PCA Memory Usage (bytes): 10107
PCA Running Time (seconds): 143
```

References

[1] CSE 220: Data Structures. *Introduction to Graphs*, April 4th, 2012.
http://moodle.bracu.ac.bd/mod/page/view.php?id=526

[2] CSC 148H. *Graphs*, April 4th, 2012.
http://www.cs.toronto.edu/ rdanek/csc148h08/assignments/a4/A4.html

[3] Wikipedia, The Free Encyclopedia. *Graph (mathematics)*, April 4th, 2012.
http://en.wikipedia.org/wiki/Graph(mathematics)