Iowa State University
Digital Repository

Graduate Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2018

Graph compression using heuristic-based reordering

Pavithra Rajarathinam
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Sciences Commons

Recommended Citation
Rajarathinam, Pavithra, "Graph compression using heuristic-based reordering" (2018). *Graduate Theses and Dissertations*. 16659.
https://lib.dr.iastate.edu/etd/16659

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Graph compression using heuristic-based reordering

by

Pavithra Rajarathinam

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Pavan Aduri, Major Professor
Samik Basu
David Fernndez-Baca

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

Copyright © Pavithra Rajarathinam, 2018. All rights reserved.

TABLE OF CONTENTS

Page

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGMENTS vi

ABSTRACT vii

CHAPTER 1. INTRODUCTION 1

1.1 Background 3

1.2 Contribution 5

1.3 Organization 5

CHAPTER 2. REVIEW OF LITERATURE 6

2.1 Index Compression 6

2.1.1 Traversal based Algorithms 6

2.1.2 Clustering based Algorithms 7

2.1.3 Encoding 7

2.2 Graph compression 7

2.2.1 Distributed partitioning 8

2.2.2 Recursive Bisection 8

CHAPTER 3. EFFECTIVE DEGREE HEURISTIC-BASED ALGORITHMS 10

3.1 Effective Degree Based Traversal (EDBT) 10

3.2 Variation of EDBT 12

3.2.1 Complexity analysis 12

3.3 Optimality Analysis 12

CHAPTER 4. EXPERIMENTAL RESULTS 15

4.1 Data sets 15

4.2 Algorithms 16

4.3 Unary Cost Analysis 17

4.4 Binary Cost Analysis 17

4.5 Permutation Analysis 18

CHAPTER 5. SUMMARY AND DISCUSSION 21

REFERENCES 22

LIST OF TABLES

Page

Table 4.1 Data Set - Graph properties 17

Table 4.2 Reordering cost 19

LIST OF FIGURES

Page

Figure 1.1 Original Graph 4

Figure 1.2 Original Index 4

Figure 1.3 Reordered Graph 4

Figure 1.4 Reordered Index 4

Figure 3.1 EDBT reordering 14

Figure 3.2 Optimal reordering 14

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Pavan Aduri for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement during tough times have inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts to review this work: Dr. Samik Basu and Dr. David Fernandez Baca I would additionally like to thank Dr. Basu for his guidance throughout the initial stages of my graduate career and Dr. Baca for his inspirational teaching style.

ABSTRACT

Inverted index has been extensively used in Information retrieval systems for document related queries. We consider the generic case of graph storage using Inverted Index and compressing them using this format. Graph compression by reordering has been done using traversal and clustering based techniques. In generic methods, the graph is reordered to arrive at new identifiers for the vertices. The reordered graph is then encoded using an encoding format. The reordering to achieve maximal compression is a well known NP complete problem, the Optimal Linear Arrangement.

Our work focuses on the inverted index format, where each node has its corresponding list of neighbours. We propose a heuristic based graph reordering, using the property that the cost of each vertex is bound by its neighbour with largest vertex id. Consider, two vertices x and y with edges a and b respectively. If x>y and a>b, then cost of graph would come down, if the vertex id of x and y are interchanged. Further, experiments shows that using this heuristic helps in achieving compression rates on par with distributed methods but with reduced utilization of computation resources.

CHAPTER 1. INTRODUCTION

Information Retrieval system maintain documents and handles search queries on the document. A typical search query is a set of key words which is used to search for documents most relevant to the query. Inverted Index has been used as in-memory data structure in Information retrieval(IR) system. The terms(words) in each document is extracted and assigned a term-ID . Each term has a corresponding list of document-ID in which they occur. The inverted index has the collection of list for all the unique terms in the document collection. Example, a term t has a list associated with it as

$$list(t) = \{d1, d2, d3\}.$$

For a given set of search terms, say $t1, t2...tn$, their document list, $list(t1), list(t2), ..., list(tn)$ is retrieved from Inverted Index and processed to get the most relevant documents. This format is efficient as it allows, processing of union or intersection of the list for a set of terms. The list often also contains the frequency of terms in each of those documents. This data structure is also used in finding group of similar documents or finding the piracy of documents.

Search engines generally get thousands of queries per second and maintain billions of documents. The inverted Index is often large and cannot be loaded completely into main memory. However, the engines have to process and return the results within few milliseconds. If the inverted index is stored in the disk and each list is loaded in memory on need basis, then the decompression can be performed for the list alone. This calls for compression at the list level in the Inverted Index to improve the processing speed. Thus, Inverted Index has the benefit of providing localized compression decompression, thereby optimizing the processing speed. Compressing the inverted index has been proven to optimize the processing of queries by Witten Et Al(11).

The inverted index can made more memory efficient by using gaps between the document id. For example, a term x with $list(x) = \{d_1, d_2, ...d_n\}$ is sorted by doc id and represented as difference

between consecutive identifiers $d_1, d_2 - d_1, ...d_{n-1} - d_n$. An example gap representation is given below,

$$list(x) : 200, 240, 300, 320, 340$$

$$list(x) : 200, 40, 60, 20, 20$$

The gaps can further be encoded and stored as binary values, thus memory bounds for a gap n is reduced to the $O(1 + log_2(n))$. This bound denotes the number of bits used to store a gap. If similar documents are assigned consecutive identifiers, then the gaps for the list corresponding to common terms would be smaller. This reduces the in-memory graph space and in turn improves the compression achieved. This is especially helpful in case of huge graphs with billions of edges. For the same list(x), if the document ids are $210, 211, 212, 213, 214$, the gaps are reduced to $list(x)$: $210, 1, 1, 1, 1$. Since the Gap compression depends on document Id's, we analyze whether reordering the document id helps with Inverted Index compression.

Additionally, this methodology can be used for social and web graph processing as well. In case of a graph, it would be similar to Adjacency list with each node having its corresponding list of neighbours. Thus graph can be represented as a collection of list for each node. Additionally, adjacency list is commonly used format for graph processing. In case of large graphs, the index can be stored in the disk and the required lists for a node can be retrieved using offset information. After retrieval, the decompression or reverting the gaps to node id can be done for required list alone which improves the decompression speed. The document and words usage of Inverted Index could be considered as a special case of graph: A bipartite graph with words and documents as node groups.

If the related nodes are reordered in a way that the node IDs are closer, then the part of the graph loaded in memory can cater successive queries. This helps to improve processing speed of the graph by having fewer cache misses. This has been extensively used in social graph processing to find similarity between user profiles and behaviour.

Generic graph compression can be achieved using various methods. In this work, we focus on reordering the vertices to reduce the gaps across the entire graph. The graph compression can be

done in two stages: First step would be reordering the vertices to reduce the gaps and second would be to use suitable encoding of the inverted index to improve level of compression. In our work, we analyze the effect of a new heuristic on the compression of graph using the inverted index format.

1.1 Background

Graph compression by reordering can been formulated and analyzed as the Optimal Linear Arrangement(OLA) or Minimum Linear Arrangement (MLA) problem. For a graph G(V,E), the Minimum Linear Arrangement is a permutation of vertices $f : V \to 1...|V|$, such that the function

$$\sum_{(u,v)\in E} |f(u) - f(v)|$$

is minimized. Its logarthmic variation (MLogA) has a corresponding minimization function as,

$$\sum_{(u,v)\in E} log|f(u) - f(v)|$$

The MLA and its logarithmic variation (MLogA) are NP- hard. If we consider the Inverted Index version of the problem, the Minimum Logarthmic Gap Arrangement(MLogGapA) [2] has been proved to be NP-hard as well. MLogGapA can be defined as, finding a permutation $f : V \to 1...|V|$ so that the $BinaryCost_f(G)$ is minimized. $BinaryCost_f(G)$ is defined as

$$BinaryCost_f(G) = \sum_{v\in V} |BinaryCost_f(v)|$$

$$where \quad BinaryCost_f(v) = \sum_{i=1}^{k-1} log|f(v_{i+1}) - f(v_i)|$$

with k neighbours of each vertex being ordered, such that for a vertex v with $neig(v) = \{v1, v2, ..., vk\}$ $f(v1) < f(v2)$. In the figure 1.1 the list corresponding to vertex 2 is $3, 4, 5$ and the gaps would be $3 : 1, 1$. Hence the $BinaryCost(2) = log(1) + log(1)$.

In case of unary encoding, the $list(x) = \{v1, v2, ..., v3\}$ is encoded as $x : v1, v2 - v1, v3 - v2$. Hence, the memory bounds of x is given by $v3$ and in generic case the neighbour with largest id. It can be noticed that the unary cost is a variation of the MLA. Following this we define the unary

cost problem as, finding a permutation of vertices $f : V \rightarrow 1...|V|$, such that the $UnaryCost_f(G)$ is minimized. $UnaryCost_f(G)$ is defined as

$$UnaryCost_f(G) = \sum_{v \in V} UnaryCost_f(v)$$

$$where \quad UnaryCost_f(v) = Max(\forall_{x \in neigh(v)} f(x))$$

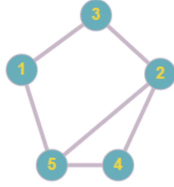

Figure 1.1 Original Graph

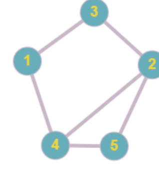

Figure 1.3 Reordered Graph

Id	Adjacency List
1	3,5
2	3,4,5
3	1,2
4	2,5
5	1,2,4

Figure 1.2 Original Index

Id	Adjacency List
1	3,4
2	3,4,5
3	1,2
4	1,2,5
5	2,4

Figure 1.4 Reordered Index

Reordering the vertices , with the objective of reducing gaps, has been studied using sequential and distributed methods. In the sequential methods, the graph is traversed based on a strategy and renamed by the order of traversal. The distributed methods are aimed at utilizing the distributed systems and enables huge graphs to be processed with lesser time. In the distributed methods, the graph is split in a top down manner such that the partitions have minimum edges between them. The partitioning ends after they reach relatively smaller size and the vertices are named at this stage. The distributed methods are especially effective for graphs with hundreds of billions edges, which cannot be processed in a single machine. The graph is partitioned and can be used in a distributed manner with minimal cross machine queries.

The graph in the figures 1.1 and 1.3 shows the Inverted Index for the original and reordered graph. The MLogGapA of the graphs are 1.67 and 1.8. It is also shown that optimal solutions for MLA and MLogA vary in graphs.(6).

1.2 Contribution

From figures 1.2 and 1.4, the highest neighbours of nodes {1,2,3,4,5} are { 5,5,2,5,4} and {4,5,2,5,4} respectively for the same graph with different ordering. The unary cost, i.e sum of list mentioned earlier, are 21 and 20. It is evident that if the vertex with highest id can be assigned to a vertex with lesser degree the unary cost comes down. It can also be noticed that if 4 is swapped with 3 the cost decreases further. We use this property in terms of effective degree, as a heuristic to reorder the graph. The effective degree can be defined as the number of vertices to which the current vertex could become the highest neighbour. In case of vertices with same effective degree, we break the ties using original degree or cost. Further modification to this heuristic provides better MLogGapA cost for the graphs analyzed. We compare the effectiveness of using this heuristic to the existing distributed work.

1.3 Organization

The rest of the thesis is organized as follows. In Section 2, we look at the existing traditional approaches to reorder the inverted index. In Section 3,we discuss our algorithm which yields better unary cost. Then we look at variations of the algorithm and the corresponding binary cost. In Section 4, we present our experimental results performed on real world graphs and comparison of our results to existing methods. In Section 5, we summarize our contributions, and discuss the possible extensions to this work.

CHAPTER 2. REVIEW OF LITERATURE

Prior related work falls into two major categories, namely compressing indices and compressing Web graphs.

2.1 Index Compression

Compressing indices has been studied based on several aspects. The stages where it can be compressed would be ordering of the vertices and the encoding of the gaps. In terms of ordering, the nodes are traversed using a strategy and then named 1 to n based on the order of traversal. The renaming of vertices is aimed at improving the compression of the inverted index.

2.1.1 Traversal based Algorithms

Modified TSP: A modified Travelling Salesman Problem has been used to find a path such that gaps between the nodes are minimized in Document Similarity Graph(DSG) (9).The DSG has documents as nodes and the number of similar terms between two documents as edge distance. Given a graph with a set of nodes and distances between the connected nodes, TSP finds the path that covers all the nodes which minimizes the distance travelled. The greedy strategy of TSP has two variation. One selects the next node as, the one with minimum distance from last node in the current path. The second method selects the next node based on node having minimum distance from any node in the current path. The second strategy studied here is based on finding the minimum spanning tree(MST) and traversing it using BFS(Breadth First Travesal) or DFS(Depth First Traversal). This is following the result that all edges of a MST are near optimal path for TSP. This has further been studied with methods to scale the algorithm using dimension reduction (8) and modified graph using Locality Sensitive Hashing (10)

Lexicographic order: In another work (5), assigning the ID based on the lexicographic order

of the document URL has proven to be much effective than other methods. But this cannot be applied to documents which lack the domain relevance property.

2.1.2 Clustering based Algorithms

Cosine Similarity based: Clustering based reordering for documents have high space complexity as it requires multiple copies of the subgraph to be maintained. The work by Blandford and Belloch [(9)], uses cosine similarity to cluster documents. Given a graph of documents, a subgraph is taken and the cosine similarity between the documents is given as edge distance. The subgraph is partitioned into two and center of mass for both are identified as D1 and D2. Each node x in the subgraph is analyzed and interchanged, such that the nodes x will be placed in partition 1 or 2 based on its highest cosine similarity to D1 or D2 respectively. This is done recursively using top down manner for the entire collection of documents. Finally, the partitions are ordered from the lowest level.Further, this has been improvised based on top down approach to utilize lexicographic property and bottom up strategy which provides linear space and time complexity.

2.1.3 Encoding

For a term x with documents, say x: 20 30 40 50, the gaps are represented as x: 20, 10,10,10. This is called as delta encoding and forms the basis for other encoding methods. We analyze the compression in terms of delta encoding with variable length byte code and BV encoding(16). The BV encoding uses a set of rules for encoding. It utilizes nodes with similar neighbours to represent the list. The BV is a widely used encoding scheme due to the availability of open source library and its performance. The Back Link encoding scheme built on the BV utilizes the shingle ordering (similarity of the list between two nodes)(6).

2.2 Graph compression

In most of the webgraph and social graph compression, the compression is based on just the nodes and the edges. These methods does not go into the detail of the nodes and the domain

specific metadata (like similarity of users or URLs). Initial work proposed in this direction would be the reordering of graph vertices using breadth first traversal(2). Recent research for partitioning or compressing graphs have been done using distributed systems. This is mainly because of the availability of computation resources and need to handle terabytes of data. Thus, graph processing had been studied using distributed systems such as Pregel(7) and Apache giraph(21)

2.2.1 Distributed partitioning

In the distributed methods, the graph is partitioned into clusters which reduces the cross cluster edges. This is mostly done by trying to find two subgraph with minimum cut. The graph is recursively divided to identify clusters and then renamed from the lowest level of clusters.

Slash burn: Graph reordering based on density has also proven to be effective. The high density subgraphs (hubs) will be identified and removed. The remaining graph will be named with highest ids. The hub identification is done recursively in a top down manner and then the vertices are renamed.

Graph embedding: Hansen showed that embedding the vertices of a graph on to a plane (3) and partitioning based on the embedding gives an O($\sqrt{logn}$) approximation for balanced partitioning algorithms. Further, embedding the vertices on to a line (17) and partitioning has been shown to optimize the graph reordering in terms of MLA.

2.2.2 Recursive Bisection

The most recent work on improving the LogGapA cost of a graph uses a variation of Kernighan-Lin heuristic, Recursive Bisection(21), to partition the graph and reorder the vertices. A graph is divided into two set of vertices V1 and V2. The vertices in each set are given a cost, which signifies the cost it contributes towards compression if moved to the other set. The vertices are sorted in the descending order of move gains.Then, each vertex in the V1 and V2 are swapped until the collective gain becomes negative. This process is repeated until there are no more vertices are swapped or an iteration limit is reached.

Algorithm 1: Recursive Bisection

```
   Data: Graph G = (V, E)
   Result: Partition G1, G2
 1 Partition V into V1 and V2;
 2 repeat
 3     for v in V do
 4         gains[v] = computeMoveGain(v);
 5     end for
 6     for v  S1, u  S2 do
 7         if gains[v] + gains[u] 0 then then
 8             exchange v and u in the sets;
 9         else
10             break ;
11         end if
12     end for
13     S1 sorted V1 in descending order of gains;
14     S2 sorted V2 in descending order of gains;
15 until not converged or iterations limit exceeded;
```

CHAPTER 3. EFFECTIVE DEGREE HEURISTIC-BASED ALGORITHMS

In this section, we discuss our algorithm and variations of it which has been used to study the unary and binary cost reduction.

3.1 Effective Degree Based Traversal (EDBT)

Intuition: A given graph ordering can be optimized by swapping the node id of two vertices say x, y, if $x > y$ and $x.degree$ $>y.degree$. We can swap the vertex id of x and y and decrease the unary cost of the graph.

Description: We use this factor in terms of effective degree as priority to traverse the graph. Effective degree is the measure of the number of vertices the node x will affect if given an id. Effective degree can be defined by a k- partial order function $f : V \rightarrow V$ such that $f^{-1}(n), f^{-1}(n-1), ..., f^{-1}(n-k)$ are defined. Given a partial order for k vertices from $n, n-1, ...n-k$ the neighbours of $f^{-1}(n)$ gets UnaryCost of n. Initially a node x with y neighbors when assigned an ID n, n forms the upper bound for each of the y nodes. For all the vertices v in the graph, we intialize $UnaryCost(v) = 0$. As the algorithm propagates the k partial order determines the cost of some vertices. Given the k-partial order, the effective degree can be defined as

$$\texttt{effectiveDegree}(v) = \sum_{\substack{x \in neigh(v) \\ UnaryCost(x)==0}} 1$$

If one of the neighbors already has a higher id neighbor, the effective degree of x is decremented and so on. This algorithm aims at improving the unary cost and is detailed in the 2. A variation of our algorithm also achieves near optimal RB binary cost.

As the graph is processed, most vertices will have similar effective degree. We try to use UnaryCost, original degree and combination of both as tie breakers.

Algorithm 2: Effective Degree Based Traversal

```
   Input  : Graph G(V,E)
   Output: Reordered graph F
1  Load the Graph into InvIndex
2  Insert the nodes into priority queue pq based on degree of a node
3  Initialize currNodeID to |v|
4  while pq is not empty do
5  |  currNode = pq.front();
6  |  Assign currNodeID to node;
7  |  Decrement currNodeID;
8  |  for Neighbors of currNode do
9  |  |  if Neighbors.UnaryCost == 0 then
10 |  |  |  for NeighOfNeigh : Neighbors.neighbors do
11 |  |  |  |  Decrement priority of NeighOfNeigh;
12 |  |  |  end for
13 |  |  |  Update Neighbors.UnaryCost = currNodeID+1;
14 |  |  end if
15 |  end for
16 end while
```

Priority processing: Often, the effective degree of the graph is similar for more than one vertex. The use of various tie breakers is essential in bringing down the unary cost. If the effective degree is similar, then cost or original degree of a node (the highest neighbour id) is used to determine the priority 9 and 9. A node with higher unary cost/original degree is given priority over the other.

Algorithm 3: Cost as tie breaker

```
1  Function Priority_Queue::compare(x, y):
2  |  if x.priority > y.priority then
3  |  |  /* y gets priority */
4  |  |  return 1;
5  |  else if x.priority == y.priority and x.cost > y.cost then
6  |  |  /* y gets priority */
7  |  |  return 1;
8  |  /* x gets priority */
9  |  return -1;
```

Algorithm 4: Original degree as tie breaker

```
1 Function Priority_Queue::compare(x, y):
2     if x.priority > y.priority then
3         /* y gets priority */
4         return 1;
5     else if x.priority == y.priority and x.originalDegree > y.originalDegree then
6         /* y gets priority */
7         return 1;
8     /* x gets priority */
9     return -1;
```

3.2 Variation of EDBT

Experiments shows that a variation of EDBT 5 provides better binary cost. The priority is decremented for each neighbour and their cost updated to recently assigned neighbour rather than the highest neighbour.

3.2.1 Complexity analysis

The algorithm iterates over v vertices and instantiates the priority with O(V). Then the neighbour are iterated and if not processed already its neighbours are iterated O(VE). The priority queue insertion takes log n time and it is the only factor which forms a bottle neck. For consecutive runs, we maintain a copy of the priority queue items. The items are thereon added to the priority queue and not modified to reduce the cost of deletion. This gives additional run time complexity of O(V log V) The run time is controlled by the edge with total cost of O(VE+V log V) and space complexity of O(V) where a copy of nodes are created for priority queue and new ordering.

3.3 Optimality Analysis

The heuristic used utilizes the property of least degree and assigned neighbour id to assign the next neighbour. This will not provide optimal cost for the cases where similar nodes are more than the effective degree. In the figure 3.1, the node with degree 1 is assigned 10, thus one node has unary cost of 1. But this leads to a unary cost greater than 10+9+9 . The actual unary cost is

Algorithm 5: Effective Degree Based Traversal for Binary Cost

```
   Input  : Graph G(V,E)
   Output: Reordered graph F
 1 Load the Graph into InvIndex
 2 Insert the nodes into priority queue pq based on degree of a node
 3 Initialize currNodeID to |v|
 4 while pq is not empty do
 5 |  currNode = pq.front();
 6 |  Assign currNodeID to node;
 7 |  Decrement currNodeID;
 8 |  for Neighbors of currNode do
 9 |  |  Decrement priority of Neighors;
10 |  |  Update Neighbors.UnaryCost = currNodeID+1;
11 |  |  if Neighbors.UnaryCost == 0 then
12 |  |  |  for NeighOfNeigh : Neighbors.neighbors do
13 |  |  |  |  Decrement priority of NeighOfNeigh;
14 |  |  |  end for
15 |  |  end if
16 |  end for
17 end while
```

Algorithm 6: Combination tie breaker

```
 1 Function Priority_Queue::compare(x, y):
 2 |  if x.priority > y.priority then
 3 |  |  /* y gets priority */
 4 |  |  return 1;
 5 |  else if x.priority == y.priority and x.UnaryCost > y.UnaryCost then
 6 |  |  if x.originalDegree > y.originalDegree then
 7 |  |  |  /* y gets priority */
 8 |  |  |  return 1;
 9 |  /* x gets priority */
10 |  return -1;
```

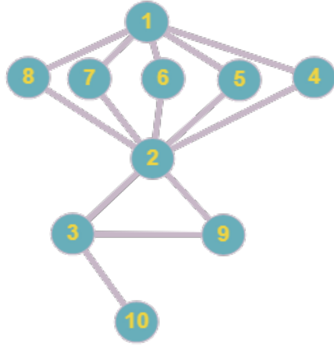

Figure 3.1 EDBT reordering

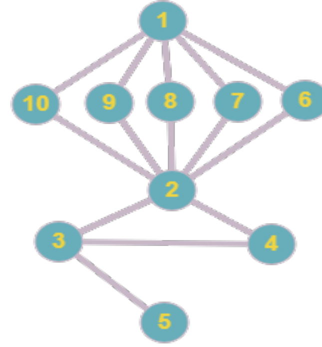

Figure 3.2 Optimal reordering

43 .In the other assignment as shown in figure 3.2, the 10 being assigned to one of the 5 similar nodes and hence 9,8,7,6 will not be the unary cost of any other node. The 10 is used to shadow the effect of 9,8,7 and 6. This brings the unary cost down to 41. Thus, in graphs with higher density, utilization of similar nodes would bring out the optimal arrangement. Optimal arrangement is dependent on the structure of the graph.

CHAPTER 4. EXPERIMENTAL RESULTS

Experiments were performed to analyze the compression achieved by reordering and encoding. by various algorithms and then compare the unary and binary cost. Further, the resulting graphs were encoded using BV compression to study the effect of reordering on the encoding. The algorithms used for traversal were BFS(Breadth First Search), Highest Degree based traversal, Effective degree with cost and tie breaker, and Effective degree with cost and tie breaker. A variation of the algorithm was found effective for binary cost and has been compared with Recursive Bisection.

Experiments were conducted on several graphs from the Stanford Large Network Dataset Collection(15). The implementation was done in Java, and run on a Linux server with Core 32GB RAM with Cent-OS operating system. The graph used for experiment are from varying node size and density. The graphs vary from 262,111 nodes (Amazon) to 1,791,489 nodes(Wiki topcats). The graph have been pre-processed to have vertices 1 to n and then traversed using one of the algorithms.

4.1 Data sets

Data sets listed below were used and their size in terms of edges and nodes are given in table 4.1

Amazon200: Nodes are Amazon items related based on categories and item purchased together. If an item i is frequently co-purchased with product j, the graph contains an undirected edge from i to j.

com-amazon.ungraph: A graph formed by crawling Amazon website. Nodes are customers and items related as who Bought an Item.

com-dblp.ungraph: DBLP Computer Science has comprehensive list of research papers in computer science. The graph is a co-authorship network where two authors are connected if they

publish at least one paper together.

com-Youtube:A graph of the Youtube social network users. Users are connected with each other and forms groups. Users can be part of a group as well

Wikitalk: Each registered Wikipedia user has a talk page, that they can edit in order to communicate and discuss updates to various articles on Wikipedia. Using the latest complete dump of Wikipedia page edit history (from January 3 2008) all user talk page changes was extracted as a graph

as-skitter: Internet topology graph. From trace routes run daily in 2005 from several scattered sources to million destinations.

cit-hepth: Its a network of citation from e-print arXiv. If a paper i cites paper j, the graph contains a directed edge from i to j. If a paper cites, or is cited by, a paper outside the dataset, the graph does not contain any information about this.

cit-Patents: A graph of the US patents which are maintained by the National Bureau of Economic Research. The data set spans 37 years (1963 to 1999), and includes all the utility patents granted during that period. The citation graph includes edges for each patent i which refers patent j within the given nodes.

4.2 Algorithms

We have used variation of the EDBT with cost and original degree as tie breaker. Similarly, for the modified version of EDBT cost and orginal degree and its combination is used as tie breaker

BFS: The graph is traversed using BFS and vertices are named according to that order

HDT: The vertices with highest degree is given the lowest ID. This can be considered as the reverse approach of EDBT

EDBT-Org: This is a variation of EDBT with original degree being the tie breaker and is assigned in the initial level only

EDBT-Cost: This is a variation of EDBT with cost being the tie breaker. The cost being modified

adds to the run time complexity of the algorithm, as priority queue operations are involved.

EDBT-Comb Mod: The modified EDBT with combination of original degree and cost as tie breaker

EDBT-Cost Mod: The modified EDBT with cost as a tie breaker

RB: Recursive Bisection was ran on the given graphs and mapping was provided to us by Sergey Pupeyrev [(21)]

Table 4.1 Data Set - Graph properties

Graph	**Node**	**Edges**
Amazon200	262,111	1,234,877
com-amazon.ungraph	334,863	925,872
com-dblp.ungraph	317,080	1,049,866
com-Youtube	1,134,890	2,987,624
wikitalk	2,394,385	5,021,410
as-skitter	1,696,415	11,095,298
cit-hepth	27,770	352,807
cit-Patents	3,774,768	16,518,948

4.3 Unary Cost Analysis

The graphs were reordered and from the mapping, the unary cost was derived as sum of highest ID neighbour for each node. The EDBT with cost as tie breaker performs better than other algorithm for all the graphs. It can also be seen that unary cost and the BV metrics(Binary cost) are not varying in the same way in any of the graphs. Another factor to notice is that, denser the graph the unary cost comes down more for the given heuristic. Thus, this method will be more effective for denser graphs.

4.4 Binary Cost Analysis

Using the reordered graph from the algorithms, we used two encoding scheme to analyze the binary cost. The delta encoding where gaps were encoded by variable byte encoding and BV

compression scheme.For the first scheme, each list is converted into gaps and then the entire inverted index is encoded as a byte stream with special byte codecs to identify each list. In terms of BV encoding, we use the mapping to arrive at the modified graph. The graph is printed out into list of edges and then the file is fed into the BV compression algorithm. The BV scheme, on processing a graph, stores it using three files: the properties file, offset file and .graph file. The properties file has meta data of the compressed graph such as bits per link and bits per interval. The binary cost varies independently of the unary cost and does not seem to be affected by the density. The file size for delta encoding shows that EDBT-comb and RB are closer and sometimes EDBT-comb yields lower size. For amazon200 and com-amazon.ungraph , the EDBT variations have lower file size. Thus, binary cost is varying based on the encoding and EDBT is more suitable for delta encoding while RB is suitable for BV compression scheme.

4.5 Permutation Analysis

We analyzed the L1 distance, L2 distance and Cosine Similarity between the permutation for all the algorithms. The results showed that permutations were random and does not follow any similarity range between the mentioned algorithms.

Table 4.2 Reordering cost

Algorithm	**Delta Encoding(KB)**	**BV(Bits per link)**	**Unary-Cost**
amazon200			
original	4,243	14.947	50,922,737,794
bfs	3,937	12.031	41,458,074,695
HDT	4,788	20.799	50,922,737,794
EDBT-Org	3,547	10.809	42,781,208,937
EDBT-cost	3,909	14.062	**40,612,996,699**
EDBT-Comb Mod	**3,188**	8.645	37,026,723,279
EDBT-Cost Mod	**3,239**	9.068	37,005,229,976
RB	**3,214**	8.747	42,003,649,153
com-dblp.ungraph			
original	5,275	16.666	75,800,000,000
bfs	4,431	10.273	52,500,000,000
HDT	5,097	18.093	47,500,000,000
EDBT-Org	4,654	13.502	76,400,000,000
EDBT-cost	4,650	14.058	**44,200,000,000**
EDBT-Comb Mod	**4,043**	9.209	56,200,000,000
EDBT-Cost Mod	**4,138**	9.581	56,500,000,000
RB	**4,031**	7.800	59,700,000,000
com-youtube-ungraph			
original	13,967	15.027	547,000,000,000
bfs	14,120	14.303	494,000,000,000
HDT	13,796	17.667	293,000,000,000
EDBT-Org	14,563	17.594	1,040,000,000,000
EDBT-cost	**13,460**	15.046	**213,000,000,000**
EDBT-Comb Mod	13,676	13.866	670,000,000,000
EDBT-Cost Mod	**13,629**	13.816	630,000,000,000
RB	**13,359**	11.578	757,000,000,000
com-amazon.ungraph			
original	4,381	23.467	146,000,000,000
bfs	4,148	10.981	60,900,000,000
HDT	4,918	21.876	58,200,000,000
EDBT-Org	3,755	10.149	65,300,000,000
EDBT-cost	4,437	16.030	**51,000,000,000**
EDBT-Comb Mod	**3,575**	9.178	58,000,000,000
EDBT-Cost Mod	**3,557**	9.088	58,000,000,000
RB	**3,567**	8.278	65,700,000,000

Table 4.2 (continued)

Algorithm	**Delta Encoding(KB)**	**BV(Bits per link)**	**Unary-Cost**
as-skitter			
original	41,197	9.974	1,600,000,000,000
bfs	40,966	10.118	1,596,137,980,366
HDT	43,253	13.731	1,062,633,411,384
EDBT-Org	44,280	13.604	2,390,000,000,000
EDBT-cost	40,880	11.403	**775,000,000,000**
EDBT-Comb Mod	**36,613**	8.533	1,680,000,000,000
EDBT-Cost Mod	**37,235**	8.928	1,770,000,000,000
RB	**35,923**	6.347	1,520,000,000,000
cit-Hepth			
original	1384	29.648	264,000,000,000
bfs	1024	8.814	504,000,000
HDT	1099	11.563	517,000,000
EDBT-Org	1057	9.989	592,000,000
EDBT-cost	1027	9.868	**470,000,000**
EDBT-Comb	969	8.391	520,000,000
EDBT-Cost Mod	972	8.374	526,000,000
cit-Patents			
original	101590	24.533	19,924,870,590,408
bfs	81317	16.742	8,209,542,343,666
HDT	96495	24.203	7,144,918,712,484
EDBT-Org	86313	19.7	10,559,104,597,943
EDBT-cost	90043	21.087	**5,954,512,341,018**
EDBT-Comb Mod	74907	15.933	8,392,018,241,308
EDBT-Cost Mod	75021	15.742	8,744,689,350,021

CHAPTER 5. SUMMARY AND DISCUSSION

We have defined a variation of MLA problem,the unary cost for a graph and analyzed the compression of graph in terms of unary and binary cost. The heuristic used here gives best unary cost and near optimal binary cost with relatively less use of computational resources. From Table 4.2, it is evident that the algorithm provides lower unary cost than the traditional methods like BFS and Highest degree first based reordering. Based on the analysis in 3.1 and 3.2, it can be seen that similarity measure can yield better compression for the mentioned cases. Hence, this heuristic if used as a combination with similarity measure would yield better compression.Further, distributed algorithms when combined with heuristic based traversal at subgraph level to reduce the iteration could benefit from the reduced space complexity.

REFERENCES

[1] Moses Charikar, Mohammad Taghi Hajiaghayi, Howard Karloff, and Satish Rao. 2006. l_2^2 spreading metrics for vertex ordering problems. *In Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA '06)*

[2] Apostolico and G. Drovandi. Graph compression by BFS. *Algorithms*, *2(3):1031-1044,2009*

[3] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM Volume 56 Issue 2, April 2009*

[4] R.Bar-Yehuda, G.Even, J.Feldman, J. Naor. Computing an Optimal Orientation of a Balanced Decomposition Tree for Linear Arrangement Problems. *J. Graph Algorithms Appl,5, 2001*

[5] Silvestri F. (2007) Sorting Out the Document Identifier Assignment Problem. In: Amati G., Carpineto C., Romano G. (eds) Advances in Information Retrieval. *ECIR 2007*

[6] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. *In Knowledge Discovery and Data Mining, pages 219228, 2009*

[7] Kevin Aydin, MohammadHossein Bateni, Vahab Mirrokni. Distributed Balanced Partitioning via Linear Embedding *WSDM, 2016*

[8] R. Blanco and A. Barreiro. Document identifier reassignment through dimensionality reduction.*ECIR (2005)*

[9] D. Blandford and G. Blelloch. Index compression through document reordering. *DCC. (2005)*

[10] S. Ding, J. Attenberg, and T. Suel. Scalable techniques for document identifier assignment in inverted indexes. *WWW (2010)*

[11] I. H. Witten, A. Moffat, and T. C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images.1994.

[12] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. *World Wide Web Conference, April 2008*

[13] M. D. Hansen. Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems. *Foundations of Computer Science, 1989., 30th Annual Symposium*

[14] Wann-Yun Shieh a, Tien-Fu Chen b, Jean Jyh-Jiun Shann a, Chung-Ping Chung. Inverted file compression through document identifier reassignment. *Information Processing and Management Volume 39, Issue 1, January 2003, Pages 117-131*

[15] SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[16] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. *In World Wide Web, pages 595602, 2004*

[17] Kevin Aydin, MohammadHossein Bateni, Vahab Mirrokni. Distributed Balanced Partitioning via Linear Embedding. *WSDM. 2016*

[18] K. H. Randall, R. Stata, R. G. Wickremesinghe, and J. L. Wiener. The link database: Fast access to graphs of the web. *In Data Compression Conference, pages 122131, 2002*

[19] Fabrizio Silvestri, Salvatore Orlando, Raffaele Perego. Assigning Identifiers to Documents to Enhance the Clustering Property of Fulltext Indexes. *SIGIR. 2004*

[20] Xu, X., Pan, S., Wan, J. (2010). Compression of Inverted Index for Comprehensive Performance Evaluation in Lucene. *2010 Third International Joint Conference on Computational Science and Optimization, 1, 382-386.*

[21] Dhulipala, L., Kabiljo, I., Karrer, B., Ottaviano, G., Pupyrev, S., Shalita, A. Compressing Graphs and Indexes with Recursive Graph Bisection. *KDD. 2016*