NOVEL HEURISTIC APPROACHES TO ITERATIVE GRAPH DRAWING

Farshad Ghassemi Toosi

Faculty of Science & Engineering
Department of Computer Science & Information Systems
University of Limerick

Submitted to the University of Limerick for the degree of
Doctor of Philosophy (PhD)

November 2017
Abstract

This thesis presents a new graph drawing approach (the concentric approach) for drawing any undirected graph without taking stress or forces into account, in order to address the trade-off between quality and running time. This approach, unlike the widely-used force-directed graph drawing algorithms, does not need to calculate the Cartesian distances between pairs of nodes in order to calculate forces or stresses. Also, the concentric approach does not need to calculate eigenvalues or eigenvectors of any matrix as is required in the state-of-the-art spectral layout algorithms. Our approach is, at the same time, similar to the class of force-directed algorithms in the sense that it is an iterative-scheme approach that updates node positions at each iteration.

The proposed approach incorporates two independent models in order to deal with two different graph representations: the adjacency matrix and the distance matrix. A distance matrix is easily extensible to a similarity/dissimilarity matrix, so that multidimensional data can be visualized as well. We compare each of our graph drawing models with the algorithms most similar to them and present our results (in terms of both running time and visual layouts) in order to evaluate the performance improvement. This is found to be significant in both cases. Our running time improvement is in the order of 5:1 for the two graph drawings models, while the layout improvement is measured qualitatively.

We also propose the use of pre-processing evolutionary algorithms in order to improve the quality of some specific types of small graphs when drawn by any iterative graph drawing algorithm, including the ones discussed in this thesis. The use of these algorithms, is shown to demonstrate a significant improvement of the quality of some specific types of graphs.

The overall conclusion of this thesis is, first of all, that our concentric approach can successfully deal with the trade-off between quality and running time. Secondly, we propose several pre-processing methods which significantly improve the layout quality for some specific types of graphs when drawn by any iterative graph drawing algorithm.
Declaration

I herewith declare that this thesis is completely my own work. I have created the thesis without making use of any help from anyone other those mentioned. This thesis has not been published, presented and submitted in order to get a similar degree to any other examination board.

This research started from AY 2012/13 to AY 2016/17 under supervision of Dr. Nikola Nikolov and joint supervision of Dr. Malachy Eaton.

Farshad Ghassemi Toosi
November 2017
I would like to thank the Department of Computer Science and Information Systems (CSIS) which has generously paid my tuition fees for three years. I would also like to thank the Irish Research Council for Science, Engineering and Technology (IRCSET) for their one-year PhD scholarship. And finally I would like to thank Plassey Campus Centre (PCC) for their two-year PhD scholarship. I am forever indebted to my supervisor Dr. Nikola Nikolov who has guided me through this fantastic opportunity with his academic assistance. I am also indebted to Ms Annette McElligott. I could not have continued my PhD without her support as Head of Department when I began my PhD. I would especially like to thank Dr. Malachy Eaton who opened a new window for me and familiarized me with a new scientific discipline which greatly helped my career. I would like to thank Dr. Patrick Healy as I have learned a great deal from him. I would like to thank Prof. Dr. Lars Linsen and Prof. Dr. Marc-Thorsten Hütt from Jacobs University in Germany that they have helped me a lot at the beginning of my PhD. I would like to thank Prof. Tiziana Margaria, Head of CSIS, for her support especially with my conference travel costs.

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

I would like to thank Prof. Paul McKevitt as my external examiner who has given me valuable comments on my Thesis.

A special thank also to Dr. Norah Power, Dr. Dermot Shinners-Kennedy, Dr. Atif Azad, Dr. Jim Buckley. I have learned a lot from them.

I would like to thank Clem O’Donnell, Gemma Swift, Marry Hacket, Nuala Kitson, Joan Imray, Liam O’Riordan, Redmond O’Brien, Eimhear O’Brien, Dr. Andrew Ju, Dr. Ali Sheikh, Azalden Alakrot, Dr. Ali Faqeh, Dr. Arash Joorabchi, Colm McGettrick, Anne Mary Hayes and Austin Dowling. I would like to thank all my friends here in Ireland and Persia for supporting me. Last but certainly not least I am indebted to my beloved Mother who has supported me for as long as I can remember. Even from
far away she has supported me through each day and has given me the energy to keep going and for that I cannot thank her enough.

Farshad Ghassemi Toosi
Limerick
November 2017
# Table of contents

## List of tables

XIII

## Notation

XVII

## 1 Introduction

1.1 Data Types .................................................. 3
   1.1.1 Graphs ................................................ 3
   1.1.2 Multidimensional Data ................................ 5
1.2 Research Problem ........................................... 5
   1.2.1 Quality ................................................ 6
   1.2.2 Time Efficiency ........................................ 7
1.3 Research Objectives ......................................... 7
1.4 Methodology ................................................ 8
1.5 Contributions of the Thesis ................................ 10
1.6 Thesis Outline ............................................. 11

## 2 Background and Literature Review

2.1 Mathematical Preliminaries ................................. 15
2.2 A Review of Graph Drawing Algorithms .................... 22
   2.2.1 The Class of Force-Directed Algorithms (Energy-Based Algorithms) 22
   2.2.2 Multilevel Paradigm ................................... 29
   2.2.3 Graph Drawing using Evolutionary Algorithms ...................... 32
   2.2.4 Spectral Matrix Graph Drawing Algorithms ....................... 36
   2.2.5 Hierarchical Graph Drawing Algorithms ......................... 38
   2.2.6 Other Graph Drawing Algorithms .......................... 39
   2.2.7 3D Graph Drawing ...................................... 39
2.3 High Dimensional Data Visualization ...................... 40
   2.3.1 Multidimensional Scaling ................................ 40
2.3.2 Sammon Projection ........................................ 41
2.3.3 k-nearest Neighbors Techniques for High Dimensional Data Vi-
        sualization ................................................... 41

3 Concentric Approach in Graph Drawing using Distance Matrix 43
3.1 Introduction ................................................ 43
3.2 Concentric Approach ......................................... 44
    3.2.1 General View .......................................... 44
    3.2.2 Concentric Approach in Visualization .................. 45
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance 52
    3.3.1 Introduction ........................................... 52
    3.3.2 Multilevel Adoption .................................... 54
    3.3.3 Optimization Solutions instead of the Multilevel Technique .. 55
    3.3.4 Trigonometrical Approximation of Sine and Cosine ............ 57
    3.3.5 The Concentric Theoretical Distance Algorithm (CTD) ........ 59
    3.3.6 Experimental Evaluation ................................ 67
    3.3.7 Conclusion ............................................. 84
3.4 CMD: Concentric Approach to Multidimensional Data Visualization
    using Dissimilarity Matrix ................................... 86
    3.4.1 Introduction ........................................... 86
    3.4.2 Datasets ............................................... 87
    3.4.3 Concentric-Approach Based Visualization for Multidimensional
        Data using Dissimilarity matrix ............................ 93
    3.4.4 Visual Results .......................................... 96
    3.4.5 Conclusion ............................................. 107

4 The Concentric Approach to Graph Drawing using Edge-Lists 109
4.1 Introduction ................................................ 109
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance 110
    4.2.1 Sync Algorithm ........................................ 111
    4.2.2 Burst Algorithm ....................................... 123
    4.2.3 Discussion ............................................. 127
    4.2.4 Visual Results and Statistics ........................... 137
4.3 Conclusion .................................................. 154
5 Investigation into the Reduction of Local-minima using Optimization Algorithms

5.1 Introduction ............................................................................................................. 155
5.2 Star-Shaped Graphs ............................................................................................... 158
  5.2.1 Star-Shaped Topology ...................................................................................... 160
  5.2.2 How to Generate Star-Shaped Graphs ............................................................ 161
5.3 INP: Initial Node Placement Effect ....................................................................... 164
5.4 Genetic Algorithm .................................................................................................. 168
  5.4.1 GA Techniques ............................................................................................... 169
  5.4.2 GA Initiation ................................................................................................. 171
5.5 Dist-GA: Distance Based Graph Drawing Initiation using GA .............................. 171
  5.5.1 Fitness Function ............................................................................................ 171
  5.5.2 Dist-GA Techniques ....................................................................................... 173
5.6 Ang-GA: Angular Based Graph Drawing Initiation using GA .............................. 174
  5.6.1 Fitness Function ............................................................................................ 176
  5.6.2 Ang-GA Techniques ....................................................................................... 177
  5.6.3 Functionality .................................................................................................. 179
5.7 Simulated Annealing .............................................................................................. 179
5.8 Results .................................................................................................................... 180
  5.8.1 p-value ............................................................................................................ 182
  5.8.2 Evaluation ..................................................................................................... 183
  5.8.3 Dist-GA .......................................................................................................... 184
  5.8.4 Ang-GA .......................................................................................................... 185
  5.8.5 SA using Theoretical Distance ...................................................................... 186
  5.8.6 Statistical Results .......................................................................................... 187
5.9 Conclusion .............................................................................................................. 188

6 Conclusion .................................................................................................................. 189
  6.1 Overall View ...................................................................................................... 189
  6.2 Relationship to the other Research Work ........................................................... 190
    6.2.1 Work related to the CTD Algorithm ............................................................ 191
    6.2.2 Work related to the CMD Algorithm .......................................................... 191
    6.2.3 Work related to the CBD Model ................................................................. 192
  6.3 Future Work ........................................................................................................ 195

References .................................................................................................................... 197
Appendix A 207

A.1 A list of Figures for Chapter 3 by CTD, SM and TASC 207
   A.1.1 CTD vs SM 207
   A.1.2 CTD using TASC 213
   A.1.3 CTD on Small Graphs using SINP 215

A.2 A list of Figures for Chapter 4 by CBD model applying Sync and Burst Algorithms 217
   A.2.1 CMD vs BStress 217
   A.2.2 3D visualization by CMD and BStress 226

Index 229
List of tables

3.1 Distance matrix for a graph with 3 nodes. ........................................ 48
3.2 Theoretical distance matrix for a graph with 4 nodes. ..................... 50
3.3 The time (seconds) for calculating $\sin$ and $\cos$ with two different approaches. ................................................................. 58
3.4 Stress Majorization (SM) vs. CTD. $E_{i}$ indicates the number of extra iterations when all nodes are introduced by MVN. The presented time is only for the actual CTD and SM algorithms. The time for reading the file, producing the visual layout and calculating the all pairs shortest paths are excluded from the reported time below. .......................... 83
3.5 CTD using Euclidean distances (PYG: Pythagoras equation) VS CTD using TASC (Manhattan distance). The running time is only for the actual drawing algorithm; the time for calculating the all-pairs-shortest-path is excluded. Other information is illustrated in Table 3.4 . . . . . 84
3.6 The dissimilarity matrix of a dataset containing 1600 individuals within four different clusters each cluster containing 400 individuals. The cells on diagonal line shows the range of dissimilarities between the individuals which belong to the same cluster and the other cells show the range of dissimilarities between the individuals from different clusters. Values in the table represent dissimilarity percentages. ............................ 88
3.7 Details of the dissimilarities for the first dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages. ................................. 89
3.8 Details of the dissimilarities for the second dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages. ................................. 89
3.9 Details of the dissimilarities for the third dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages. ........................................ 90

3.10 Details of the dissimilarities for the forth dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages. ........................................ 90

3.11 Details of the dissimilarities for the fifth dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages. ........................................ 90

3.12 Details of the dissimilarities for the sixth dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages. ........................................ 90

3.13 Details of the dissimilarities for the dataset used for 3D visualization with 1000 individuals and 5 different groups (200 individuals per group). Values in the table represent dissimilarity percentages. ........................................ 91

3.14 Details of the dissimilarities for the BBC news dataset with 2225 documents and 5 different groups; 1: Business, 2: Entertainment, 3: Politics, 4: Sports, 5: Tech. Values in the table represent dissimilarity percentages. ........................................ 104

3.15 Details of the dissimilarities for the BBC sport news dataset with 737 documents and 5 different groups; 1: Athletics, 2: Cricket, 3: Football, 4: Rugby, 5: Tennis. Values in the table represent dissimilarity percentages. ........................................ 105

3.16 Details of the dissimilarities for the chat data-set with 990 documents and 3 different groups; 1: Victims, 2: Predator, 3: Normal talks. Values in the table represent dissimilarity percentages. ........................................ 106

3.17 Details of the dissimilarities for the chat dataset with 1500 documents and 4 different groups; 1: Predators, 2: Victims, 3: Celebrities 4: Teenagers. Values in the table represent dissimilarity percentages. ........................................ 107

4.1 Details of the comparison running time between the model by Hu and CBD model for a few selected graphs. ........................................ 129

4.2 Details of the running time for the BStress model and the CBD model on a few selected graphs. ........................................ 135

4.3 Details of some small graphs using CBD. No.Sync and No.Burst are the number of iterations spent on sync algorithm and burst algorithm respectively; (see Algorithm 7 and Algorithm 8) ........................................ 138

4.4 Details of some large graphs using CBD. The running time of the sync algorithm and sync & burst algorithms (Total) are reported separately. 140
List of tables

5.1 The results of applying Dist-GA on star-shaped graphs for node-counts [50, 90]. ................................................................. 184
5.2 The results of applying Dist-GA on cactus graphs for node-counts [50, 90]. ................................................................. 185
5.3 The results of applying Dist-GA on random graphs for node-counts [50, 75]. ................................................................. 185
5.4 The results of applying Ang-GA on star-shaped graphs for node-counts [50, 90]. ................................................................. 185
5.5 The results of applying Ang-GA on cactus graphs for node-counts [50, 90]. ................................................................. 185
5.6 The results of applying Ang-GA on random graphs for node-counts [50, 75]. ................................................................. 186
5.7 The results of applying SA on star-shaped graphs for node-counts [50, 90]. ................................................................. 186
5.8 The results of applying SA on cactus graphs for node-counts [50, 90]. ................................................................. 186
5.9 The results of applying SA on random graphs for node-counts [50, 75]. ................................................................. 187
5.10 Statistical results of p-value with three different algorithms on three different data-sets. ................................................. 187
5.11 Statistical results of running time (Sec) with three different algorithms on three different data-sets. ................................................. 188
Notation

- **3D** refers to 3 dimensional.

- **Ang-GA** refers to the *Angular Based Graph Drawing Initiation using Genetic Algorithm*, on page 174.

- **ATD** refers to the *Angular-coordinate driven radial Tree Drawing* algorithms [114] which represents two slightly different algorithm to draw any undirected tree in a very short time, on page 168.

- **BCDD** refers to the *Betweenness Centrality Degree Distribution* of all the nodes in a given graph, on page 160.

- **BFS** refers to the *Breadth-first search* algorithm which is an algorithm to traverse graphs and it was invented in 1950 by E. F. Moore.

- **BStress** refers to the *Binary Stress Model for Graph Drawing* algorithm developed by Koren *et al* [82], on page 27.

- **Burst** refers to the second algorithm of the CBD model, on page 123.

- **cosine** refers to the *Cosine of an angle that is created by $\overrightarrow{p_{ni},p_{nj}}$ and the x axis*.

- **CBD** refers to the *Concentric Approach for Binary Distance Graph Drawing* algorithm, on page 110.

- **CMD** refers to the *Concentric method for Multidimensional Data Visualization* algorithm, on page 86.

- **CTD** refers to the *Concentric Approach for Theoretical Distance Graph Drawing* algorithm, on page 52.

- **Dist-GA** refers to a *Distance Based Graph Drawing Initiation using Genetic Algorithm*, on Section 171
• FR refers to the Fruchterman and Reingold algorithm [38], on page 23.

• INP refers to the Initial Node Placement, on page 164.

• KK refers to the Kamada and Kawai algorithm [70], on page 24.

• MDS refers to the Classical Multidimensional Scaling algorithm [67].

• MVN refers to the Multilevel Vertex Neighboring algorithm, on page 54.

• QT refers to Quad Tree, on page 114.

• R-Crs refers to the Number of Edge-Crossings when the Drawing Starts with Random Initial Node Placement, on page 183

• R.time refers to the Running Time of a Drawing when it Starts with Random Initial Node Placement, on page 183

• R.wins refers to the Number of Times that Drawing Wins when it Starts with Random Initial Node Placement, on page 183

• S-Crs refers to the Number of Edge-Crossings when the Drawing Starts with Smart Node Initial Placement, on page 183

• S.time refers to the Running Time of a Drawing when it Starts with Smart Initial Node Placement, on page 183

• S.wins refers to the Number of Times that Drawing Wins when it Starts with Smart Initial Node Placement, on page 183

• SA refers to the Simulated Annealing Algorithm, on page 179

• SDE refers to the Graph Drawing Using Spectral Distance Embedding algorithm by Civril et al [21], on page 37.

• SINP refers to the Smart Initial Node Placement algorithm, on page 55.

• sine refers to the Sine of an angle that is created by \( \overrightarrow{p_{ni}, p_{nj}} \) and the x axis.

• SM refers to the Stress Majorization algorithm developed by Gansner et al [39], on page 26.

• SSDE refers to the Fast Graph Drawing Using Sampled Spectral Distance Embedding algorithm by Civril et al [22], on page 37.
• **Sync** refers to the first algorithm of the CBD model, on page 111.

• **TASC** refers to the *Trigonometrical Approximation of Sinus and Cosinus*, on page 57.

• **t-SNE** refers to a data visualization algorithm by Maaten et al [85], on page 42.
Chapter 1

Introduction

This thesis explores the field of graph drawing or more generally information visualization. Information visualization is becoming more important with the increasing volume of data. With the high rate of data growth the need for modeling, inspecting and discovering useful information from data is becoming very crucial [53]. One of the advantages of information visualization is that it makes it easier for a person to understand the structure of the information even if he/she is not an expert in the related field. Visualization is able to provide an insight into the given data, by encoding data individuals as visual objects (e.g. small circles) and the relations between data individuals as other types of visual objects (e.g., lines, curves, or even certain amounts of space ) [12, 19, 91]. Information visualization can be a real-time and/or user-interactive approach tool for mining data visually [103]. Information visualization can be very expressive and it does not need a very knowledgeable user to interpret the data and get an insight into it. However, a low quality visualization can easily mislead the end user. Information visualization can provide an abstract model of a large dataset in an easily comprehensible way [72, 112].

The term *big data* describes a very large amount of data with a complicated, complex and raw structure, possibly from different disciplines and with many different features [102]. The trend of data increasing is continuing as the cost of storing data is getting even cheaper than ever before. As the volume of data increases, the dependencies between data components are getting more important and complicated. Therefore, the importance of maintaining and interpreting *big data* is becoming more crucial every day [132, 65].

Different disciplines have different types of data which come with different features. For example, data can be a set of Cartesian coordinates in 2D or 3D space or a set of frequencies for certain words in a document, etc. Data can be also represented in a
way to show the relations between data components. To analyze a large amount of data one needs an appropriate tool corresponding to the data being studied.

The field of information visualization is closely related to several other different fields such as computer graphics, human-computer interaction, business, data mining, economics and psychology. Its task is to create a visual and intuitive representation of a dataset in order to let users see and explore a large amount of information at once.

Ideally, it is expected to let users interact with the visual representation of the dataset in order to discover hidden patterns [57, 71]. There are multiple different visualization techniques, each suitable for a specific type of data; e.g. a scatter plot is a very simple and basic visualization technique which takes the Cartesian coordinates of a number of individual data items and visualizes them in a mathematical diagram or Cartesian space, (see Figure1.1). The Dendrogram and circle packing are two examples of visualization which is exploited when hierarchical data (a dataset which is clustered hierarchically) is the subject of visualization [124], (see Figure1.2). The heat map as another visualization technique represents values associated with data by colors in a 2D environment [93], (see Figure1.3).

Fig. 1.1 An example of a scatter plot with 17 points.
1.1 Data Types

Data is not always defined by its individuals’ values and/or type. Frequently, relationships between data individuals in a given dataset are also either an explicit, or implicit part of the dataset. Two common ways to represent the relations between individual data are as follows:

1.1.1 Graphs

A graph is a mathematical data model that typically maps pairwise relations between objects (data individuals) [129, 14], where objects represent entities of a particular type, e.g. a person, a city, a cell, etc.

In graph theory objects are known as nodes - sometimes vertices - and the relations between nodes are known as edges - sometimes links. An undirected graph \( G = (N, E) \) is a pair of a set of nodes \( N (n_1, n_2, ..., n_{|N|} \in N) \) and a set of edges \( E (e_1, e_2, ..., e_{|E|} \in E) \). Each edge \( e \in E \) represents an unordered pair of nodes \( \{n_i, n_j\} \) and it is denoted by \( e = \{n_i, n_j\} \). The set of nodes and the set of edges in each graph can be represented.
either by a square matrix, that is called adjacency matrix, in which rows and columns correspond to nodes and each cell represents the edge (or its absence) between a pair of adjacent nodes, or by a node-link diagram [43, 44, 74]. Drawing a graph as a node-link diagram can reveal the relational structure of the data in an easy form to understand especially if the graph is large and difficult to get insight to from its matrix [43]. A graph needs to be drawn in an expressive way in order to reveal the graph’s structural details. For instance in Figure 1.4 there are two layouts representing the same graph. The layout in Figure 1.4a is not clearly showing the structure of the graph (circular structure) as much as the layout in Figure 1.4b is. Note: a graph must not be confused with graph layout, a graph can have several different graph layouts [11].

Not only an expressive graph layout is an important matter, but for a very large graph the time efficiency (how fast a graph can be drawn) is almost as important as the quality of the layout. For instance, drawing a graph with around 50K nodes within several minutes might not be always desirable. On the other hand in order to draw a large graph fast the quality of the layout may suffer [39]. A trade-off between layout quality and time efficiency is something that has to be considered.

Fig. 1.3 An example of a heat map with 225 data individuals colored based on their values (with 5 discrete colours).
1.2 Research Problem

(a) A non planar layout of a graph with 26 nodes.  
(b) A planar layout of a graph with 26 nodes.

Fig. 1.4 Two different layouts for the same graph.

1.1.2 Multidimensional Data

A dataset with a large number of attributes (dimensions) is called high dimensional data [66, 17].

Individual data with similar attributes are said to be related (similar) to each other, therefore, one can derive the relations between all data individuals in order to use them for visualization using techniques such as vector space modeling and cosine similarity [86, 104, 98, 131, 106, 64]. For instance a large number of text documents is multidimensional data if the number of words’ occurrence for each different word is considered as an attribute. Figure 1.5b shows an example of a dataset which reveals 5 different clusters with data individuals in each cluster related to each other because of the similarity between their attributes.

1.2 Research Problem

Generally speaking, Graph Drawing tries to convert a given dataset as an adjacency matrix or edge list into a visual representation (node-link diagram) in order to provide useful information; e.g, details or structure of a given graph. This is achieved if and only if the visual representation improves our insight into the given dataset. That means how well a graph drawing technique can draw a graph in order to express the details of the graph in terms of given aesthetic criteria in order to make the data interpretation task easier [95, 60, 96, 94]. On the other hand, time efficiency is also an
Fig. 1.5 Two different projections for the same data with 2000 data individuals in 5
different groups. Average similarity between two groups red and green is 0.01% and
the same average similarity between cyan and pink.

important factor in graph drawing when the graph is large [54, 36, 95]. These two main
factors: Quality and Time efficiency are like two heads of a lever so that improving
one head comes at the expense of the other head. Hu [60] lists some different graph
drawing algorithms together with their visual results and running times [60, 40].

1.2.1 Quality

One of the challenges in information visualization is to create a visual representation of
data that can increase the end user’s insight into the data [73, 23]. The data individuals
of a dataset usually have several different attributes that represent the whole big data.
The visualization should be able to take care of some aesthetic criteria for the data
studied. A user can learn more details of a given graph layout when more aesthetic
criteria are considered in a visual representation. Figure 1.6 shows two layouts of a
graph with 68 nodes. Figure 1.6b shows a planar layout while Figure 1.6a shows a
non-planar layout for a planar graph. For example, edge-crossings can make it difficult
to judge whether a graph is planar or not especially if graph is large in terms of node
count.

This problem is not only found in the field of graph drawing. Multi-dimensional
data visualization might suffer from this issue as well when the visualization technique
1.3 Research Objectives

There are a multitude of different visualization techniques and a large number of researchers working in the field of information visualization. This is because the size of data is increasing every day and the need for high quality visualization with an acceptably fast running time is even more essential than before. Graph drawing techniques play a prominent role in information visualization, because nowadays more and more interrelated data is being collected (e.g., data from social media platforms, data from multiuser applications, etc.) [24, 14]. Quality and running time in the field of graph drawing are the two opposite factors in which sometimes improvement in one comes at the expense of the other [39]. In other words, generally to improve one

(a) A layout by Fruchterman and Reingold [38] with some edge-crossings

(b) A planar layout by Fruchterman and Reingold [38].

Fig. 1.6 Two different layouts by the same graph drawing algorithm on the same graph.

is not able to produce an expressive result and instead misleads the human cognition into a wrong conclusion. Figure 1.5a and Figure 1.5b show two different visualizations of the same dataset with two different visualization techniques.

1.2.2 Time Efficiency

Another important challenge in the field of information visualization (both graph drawing and multidimensional data visualization) is its time efficiency (the time needed to draw a given graph), especially for large datasets [92, 54, 22, 36, 60]. To have a real-time user-interactive visualization, visual results have to be updated quickly. Therefore running time in information visualization is another major concern of scientists especially when working with big data.

1.3 Research Objectives

There are a multitude of different visualization techniques and a large number of researchers working in the field of information visualization. This is because the size of data is increasing every day and the need for high quality visualization with an acceptably fast running time is even more essential than before. Graph drawing techniques play a prominent role in information visualization, because nowadays more and more interrelated data is being collected (e.g., data from social media platforms, data from multiuser applications, etc.) [24, 14]. Quality and running time in the field of graph drawing are the two opposite factors in which sometimes improvement in one comes at the expense of the other [39]. In other words, generally to improve one
of these factors the other one has to be sacrificed. Since these two factors are very important in graph visualization, especially when working with large graphs, a trade-off between these two factors is a key issue in graph drawing.

The main objective of this thesis is to present a new class of graph drawing algorithms that does not include some of the existing computations in the traditional and currently used graph drawing algorithms, while the quality of layouts is kept reasonably high. Our new proposed techniques are able to improve the running time compared to similar current graph drawing techniques in the literature, while keeping the quality at the same level, and for some cases better than the quality of the existing algorithms.

1.4 Methodology

In this thesis a number of novel algorithms in the field of graph visualization are introduced. Therefore the focus of this thesis is on designing and developing these new algorithms. Although there is no standard methodology for designing a new algorithm we follow a methodology which is divided into nine different steps.

1. Problem definition
   As mentioned earlier, the two major concerns in the field of information visualization (graph drawing and multidimensional data visualization) are quality and time efficiency. Typically optimizing one of them results in sacrificing the other. Our goal is to balance the trade-off between these two factors.

2. Data Gathering
   We used two main strategies for data gathering:
   
   (a) Generating random data based on some pre-defined features:
       This research mainly focuses on developing a few novel visualization algorithms in order to improve several aspects of the available algorithms in the literature. Therefore, it is very important to test the algorithms with enough data in order to validate the usefulness of the newly developed algorithms. All data which have been generated randomly are in the form of pairwise relationship matrices. The pairwise relationship is either an adjacency matrix (or edge list) which gives a binary relation between each pair of nodes (in the field of graph drawing) or it can be an indication of the dissimilarity between each pair of nodes (in the field of multidimensional data visualization).
(b) Collecting real world data or benchmarks:
Another type of data we used in this research is real world data or data which have been already used for evaluation of similar algorithms. An example of real world data is a collection of text documents with dissimilarities between all pairs of documents. Alternatively, we have also made use of a large number of graphs from a repository made by Hu [62].

3. Defining an approach
The very first step after defining the problem is introducing an approach in order to deal with the problem. Our initial idea in order to create an approach was formed based on our work (dynamics synchronization on networks) in 2012 [119]. In that work we made use of dynamical process (synchronization) in order to find highly interconnected groups of nodes and visualize them using Classical Multidimensional Scaling (CMDS) technique.

4. Development of a model
Once a general approach is defined, different models can be developed within it.

5. Specification of an algorithm
Once the model is designed and developed then we can decide about the algorithms’ specifications. For instance what type of input data the algorithm can take in, or what type of outputs the algorithm can generate.

6. Algorithm design
The design of an algorithm is the core of our work. In this research multiple algorithms are presented based on different requirements and different types of datasets. Our main algorithms are iterative scheme algorithms which means they start with an initial step and then perform a certain number of iterations.

7. Algorithm implementation and validation
Once the algorithm is designed, we need to implement it, validate it, and check its correctness.

8. Analysis of algorithm
After testing the correctness of an algorithm one needs to analyze its time complexity and further optimize it.

9. Comparative testing
The final step is to evaluate the contribution to the field of study. To do this
one needs to propose comparison between the developed algorithms and the state-of-the-art algorithms applied to the same problem.

We break down our algorithms in steps and present them as *Pseudo-Code* that are not computer codes but easy to convert into computer codes and easy for the non-programmer to understand [47].

### 1.5 Contributions of the Thesis

Some parts of this thesis have been published and presented in a number of different international conferences. The list of the published works are presented below. All these papers are directly related to this thesis, however, some modifications and optimizations were applied in order to present them all in one coherent document.

1. **Title:** Projection-based visualization of dynamical processes on networks  
   **Authors:** Farshad Ghassemi Toosi, Fernando V Paulovich, Marc-Torsten Hütt and Lars Linsen  
   **Conference Paper:** Eurovis 2012. Published and Presented (peer reviewed)

2. **Title:** Circular Tree Drawing by Simulating Network Synchronization Dynamics and Scaling  
   **Authors:** Farshad Ghassemi Toosi and Nikola S Nikolov  
   **Conference Paper:** GD 2014. Published and Presented (peer reviewed)

3. **Title:** Evolving smart initial layouts for force-directed graph drawing  
   **Authors:** Farshad Ghassemi Toosi, Nikola S Nikolov and Malachy Eaton  
   **Conference Paper:** Gecco 2015. Published and Presented (peer reviewed)

4. **Title:** Vertex-neighboring multilevel force-directed graph drawing  
   **Authors:** Farshad Ghassemi Toosi and Nikola S Nikolov  
   **Conference Paper:** SMC 2016 (IEEE). Published and Presented (peer reviewed)

5. **Title:** Angular-Coordinate Driven Radial Tree Drawing  
   **Authors:** Farshad Ghassemi Toosi and Nikola S Nikolov  
   **Conference Paper:** Waset 2013. Published and Presented (peer reviewed)

6. **Title:** Simulated Annealing as a Pre-Processing Step for Force-Directed Graph Drawing  
   **Authors:** Farshad Ghassemi Toosi, Nikola S Nikolov and Malachy Eaton  
   **Conference Paper:** Gecco 2016. Published and Presented (peer reviewed)
7. Title: A GA-Inspired Approach to the Reduction of Edge Crossings in Force-Directed Layouts  
   Authors: Farshad Ghassemi Toosi, Nikola S Nikolov and Malachy Eaton  
   Conference Paper: Gecco 2016. Published and Presented (peer reviewed)

8. Title: Sync-and-Burst: Force-Directed Graph Drawing with Uniform Force Magnitudes  
   Authors: Farshad Ghassemi Toosi and Nikola S Nikolov  
   arXiv: 2015. Published (technical report)

Two articles are ready to be submitted as the main outputs of this thesis.

1. Title: Efficient Graph Drawing Algorithm utilizing Theoretical Distance Matrices  
   Authors: Farshad Ghassemi Toosi, Nikola S Nikolov, Malachy Eaton and Azalden Alakrot  
   Ready to submit to Journal of Graph Algorithms and Applications

2. Title: A Single-Level Fast Drawing of Undirected Graphs utilizing Synchronization Approach  
   Authors: Farshad Ghassemi Toosi, Nikola S Nikolov and Malachy Eaton  
   Ready to submit to Journal of Graph Algorithms and Applications

1.6 Thesis Outline

The main research contributions of the thesis are in the field of Graph Drawing.  
A general flowchart of our work illustrating different and new techniques, approaches and algorithms is illustrated in Figure 1.7.

The structure of the thesis is as follows:

1. Chapter 2: In this chapter we review important work related to our field of research. We discuss the specifications and the results of related state-of-the-art algorithms. We explore carefully the literature review in order to find the most relevant algorithms to ours and use them as the state-of-the-art.

2. Chapter 3: In this chapter two novel visualization algorithms are introduced.
   (a) Graph drawing utilizing theoretical distance matrices.
   (b) Multidimensional data visualization utilizing dissimilarity matrices.
This chapter introduces two variations of utilizing a new multilevel framework algorithm. The results of the first algorithm in Section 3.3 are compared to state-of-the-art method stress majorization algorithm [39], and the results of the second algorithm in Section 3.4 are compared to the results of the state-of-the-art t-Distributed Stochastic Neighbor Embedding (t-SNE) technique [85], which is now the state-of-the-art in multidimensional data visualization [110] as well as classical MDS [67] technique which traditionally used for high dimensional data visualization.

3. Chapter 4: In this chapter a new graph drawing model is presented which, unlike the algorithms in Chapter 2, utilizes an adjacency matrix. This model is comprised of two algorithms. Chapter 3 introduces and discusses all the details of the two algorithms in detail. At the end of the chapter the results of these algorithms are compared to the results of the state-of-the-art (binary stress model) [82] as well as to the results of the multilevel algorithm by Hu [60] implemented in GraphViz [41].

4. Chapter 5: This chapter introduces a novel and efficient technique which improves the quality of several single-level force-directed graph drawing algorithms [38, 70] as well as the results of the algorithms proposed in Chapter 2 and Chapter 3. This technique consists of the computation of smart initial layouts for iterative
graph drawing algorithms. We show evidence of a significant improvement of quality in terms of reducing the number of edge crossings in the graph layout. Three different variations are introduced: two with different genetic algorithms and one with simulated annealing algorithm.

5. Chapter 6: In this chapter we look at the whole thesis from the top down and discuss the results of the proposed methods. We discuss the relational works to each algorithm presented in this thesis. We discuss any possible future research work resulting from this thesis.
Chapter 2

Background and Literature Review

In this chapter, we present the literature review related to this thesis. We review the terminology and the various graph/data visualization algorithms.

2.1 Mathematical Preliminaries

In this part the mathematical terminology employed in this work is presented.

Directed and Undirected Graph

1. Directed Graph
   A directed graph or digraph $D = (N, E)$ is comprised of two sets, a set of nodes $N$ and a set of edges $E$. Each edge $e \in E$ represents an ordered pair of nodes $(n_i, n_j)$.

2. Undirected Graph
   An undirected graph $G = (N, E)$ is comprised of two sets, a set of nodes $N$ and a set of edges $E$. Each edge $e \in E$ represents an unordered pair of nodes $\{n_i, n_j\}$.

Adjacent Nodes

In $G = (N, E)$, two nodes $n_i$ and $n_j$ are adjacent if there is an edge $e = \{n_i, n_j\} \in E$

Incident Edge

An unordered edge $e = \{n_i, n_j\} \in E$ is said to be incident to the node $n_i$ and node $n_j$. 
Degree of Node

The degree $\text{deg}$ of a node $n_i$ is the number of incident edges to that node:

$$\text{deg}(n_i) = \sum_j |\{\{n_i, n_j\} \in E\}$$

where $\{n_i, n_j\} \in E$ represents an edge between $n_i$ and $n_j$.

Path in Graph

Let $n_1, n_2, ..., n_k \in N, k \geq 1$ and $n_1 \neq n_k$ and each $\{n_i, n_{i+1}\} \in E$ for $i = 1, 2, ..., k - 1$; the ordered set of nodes $P = (n_1, n_2, ..., n_k)$ is called a simple path between two nodes $n_1$ and $n_k$ if $n_i \neq n_j$ for each $i \neq j$.

Note: A pair of nodes $(n_i, n_j)$ (if $\{n_i, n_j\} \notin E$) can have multiple paths between them.

Connected Graph

A graph is called a connected graph if there is at least one path between any pair of nodes.

Adjacency Matrix

A square matrix $M_{|N| \times |N|}$ is called an adjacency matrix of $G = (N, E)$ if:

$$M[i][j] = \begin{cases} 
1 & \text{if } \{n_i, n_j\} \in E \\
0 & \text{otherwise}
\end{cases}$$

Edge List

The list of all the edges $e_1, e_2, ..., e_{|E|} \in E$ in $G = (N, E)$ is called edge list.

Weighted/Unweighted Graph

A graph that has a weight or a value associated with each edge is called weighted graph. A graph that has a unit weight for all edges is called unweighted graph.

Shortest Path

In an unweighted graph, the shortest path between two nodes $n_i$ and $n_j$ is $SP(n_i, n_j)$. The number of edges in the shortest path is called: Geodesic Distance or Theoretical Distance.
2.1 Mathematical Preliminaries

Distance Matrix

A square matrix $D_{|N| \times |N|}$ is called a distance matrix of $G = (N,E)$ if:

$$D[i][j] = \begin{cases} SP(n_i, n_j) : & \text{if } (n_i \neq n_j) ; \ SP(n_i, n_j) \text{ is the shortest path between } n_i \text{ and } n_j \\ 0 : & \text{if } (n_i = n_j) \end{cases}$$

Similarity/Dissimilarity Matrix

Let $C = (n_1, n_1, \ldots, n_P)$ be a set of data individuals $P$, and let the $VC = (V_{n_1}, V_{n_2}, \ldots, V_{n_P})$ be the weight list that represents the weight (value) of each data individual from list $C$. The dissimilarity matrix of the list $C$ is defined:

$$Dis[i][j] = \{ |V_{n_i} - V_{n_j}| \}$$

and the similarity matrix of the list $C$ is defined:

$$Sim[i][j] = \{ Max - |V_{n_i} - V_{n_j}| \}$$

$Max$ is the maximum dissimilarity value that a pair of nodes ($n_i$ and $n_j$) has.

Betweenness Centrality of a Node

Let $SP(n_i, n_j)$ be the shortest path between two nodes $n_i$ and $n_j$, and let $B(n_i, n_j) = (n_1, n_2, \ldots, n_k \in N)$ be a list of nodes that occur over the shortest path between two nodes $n_i$ and $n_j$. The betweenness centrality of $n_k$ is defined thus:

$$Bt(n_k) = \sum_{j \neq i \neq k} \left( \frac{\delta(k)_{i,j}}{\delta_{i,j}} \right)$$

where $\delta(k)_{i,j}$ is the number of different shortest paths between $n_i$ and $n_j$ that pass through $n_k$ ($n_k \in B(n_i, n_j)$) and $\delta_{i,j}$ is the number of all different shortest paths between $n_i$ and $n_j$.

Betweenness centrality is a parameter for nodes and it shows how theoretically central a node is.

Graph Drawing

Graph drawing forms an intersection between mathematics and computer graphics. Typically a graph representation, e.g. edge list or adjacency matrix, is given as an input to a graph drawing algorithm which calculates the coordinates of the graph’s nodes and the shape of the edges, then the graph is visualized by using computer
graphics features, e.g. edge thickness, node size, coloring, 3D implication, shadow, and etc. The result of a graph drawing is sometimes called a graph layout (see Section 2.1).

**Cartesian Coordinates**

Cartesian coordinates specify the location of a point in a plane (two-dimensional space), or in three-dimensional space. The Cartesian coordinates of a node are a pair of numbers (in two-dimensions) or a triplet of numbers (in three-dimensions) that specify signed distances from the coordinate axis. We use \( px_n, py_n \) and \( pz_n \) to show the \( x, y \) and \( z \) coordinates of a node \( n_i \) and \( p_n \) to show the set of Cartesian coordinates for a node \( n_i \).

**Graph Layout**

For an undirected graph \( G = (N, E) \), a set of Cartesian points \( P = \{p_{n_1}, p_{n_2}, \ldots, p_{n_{|N|}}\} \) for \( |N| \) nodes, and a set of line segments \( L = \{l_1, l_2, \ldots, l_{|E|}\} \) where \( l_i = p_{n_i}p_{n_j} \) and \( \{n_i, n_j\} \in E \) comprises a graph layout \( \gamma \) of graph \( G \).

**Euclidean Distance**

Let \( \gamma \) be the layout of graph \( G \). The Euclidean distance between any two different arbitrary nodes \( n_i \) and \( n_j \) is denoted as \( d(n_i, n_j) \).

**Edge Crossing**

Let \( p_{n_1}, p_{n_2}, p_{n_j} \) and \( p_{n_2} \) be the Cartesian positions of four nodes in the layout of a graph \( G \) if \( e_1 = \{n_1, n_2\} \in E \) and \( e_2 = \{n_j, n_2\} \in E \) in the graph \( G \). Let \( l_1 = p_{n_1}p_{n_2} \) and \( l_2 = p_{n_1}p_{n_2} \) be the two line segments.

If there is a Cartesian point \( p_c \) that \( p_c \in l_1 \) and \( p_c \in l_2 \) then it is said that there is an edge crossing at edges \( e_1 \) and \( e_2 \).

**Pairwise Information**

We make use of the term so called *pairwise information* which refers to any available type of information (binary, integer or real value) associated with a pair of nodes. For instance, the pairwise information in an adjacency matrix is either 0 or 1, or the pairwise information a distance matrix with 1 unit edge length is an integer value.
2.1 Mathematical Preliminaries

**Power Law**

Power law is a function between two variables $x$ and $y$ in which a change in one variable causes a proportional change in the other variable. A power law function is illustrated as: $y = kx^\alpha$ where $y$ and $x$ are the two variables, $\alpha$ is called the power law exponent and $k$ is a constant [6, 5]. One of the applications of the power law is Barabási–Albert model [1] in which the degree distribution follows the power law, and usually $2 < \alpha < 3$.

**Iterative Methods**

An iterative method is a mathematical procedure that contains a sequence of steps to improve a given initial solution.

**Force-Directed Graph Drawing**

Force-directed graph drawing is a class of graph drawing algorithms which contains several different algorithms [38, 70, 19, 82, 15, 9, 40, 39, 122, 60]. This class of algorithms evaluates the energy of the graph which is typically the differences between opposite forces, (see Section 2.1). The objective is to minimize the energy of the system. A clear layout with less visual confusion (a layout without node-edge cluttering) has a direct relation with a minimum energy. Force-directed algorithms belong to the class of iterative methods as they start from an initial layout and improve the layout over a number of iterations.

**Attraction and Repulsion forces**

Most of the force-directed algorithms employ two different (opposite) forces, attraction and repulsion forces. Attraction forces are applied between some nodes connected by an edge, while repulsion forces is applied between each pair of nodes. An exception is the algorithm of Bannister et al. which adopts an additional third force [4].

**Aesthetic Criteria**

Aesthetic criteria, in the field of graph drawing, refer to some important factors which the graph layout is judged. One of the very important aesthetic criteria in a graph layout is the number of edge crossings in the final layout [78, 95, 19]. There are other types of aesthetic criteria such as uniform edge length, less node cluttering, and symmetry of the layout.
Local-Minima

The local-minimum of a function of \( y = f(x) \), refers to a point \( x_m \) where \( f(x_m) < f(x) \) for \( x \) within some distance \( \epsilon > 0 \) of \( x_m \). In the context of the graph drawing, \( f(x) \) can be the energy function in a force-directed algorithm [78], or the number of edge crossings.

N-body Problem

The N-body problem is the problem of calculating the movement of an individual among a set of individuals \( N \). The movement of each individual depends on the positions of all other individuals in the groups. This causes \( O(|N|^2) \) time to calculate all of the individuals’ movements. The N-body problem occurs in most of the graph drawing algorithms that need to calculate the repulsion forces between node \( n_i \) and all other nodes. The N-body problem causes high running time. Two main commonly used solutions are proposed in the literature for this problem: The multilevel paradigm and quad tree. The time complexity can be improved to \( O(|N|\log|N|) \) by utilizing quad tree.

Node Coarsening

In the field of graph theory and more specifically graph drawing, node coarsening refers to a technique where the nodes, with a high degree of interconnectivity, are grouped in order to create a coarser node; i.e. each pair of adjacent nodes is collapsed in order to become one coarser node. This process can continue until one or a few coarsest nodes remain. This technique has application in multilevel graph drawing algorithms [9].

Multilevel Paradigm

Traditional force-directed graph drawing algorithms initially start the process of drawing by positioning all of the nodes randomly on the screen and adjust their positions towards an equilibrium state of minimum energy over a certain number of iterations. This means that the applied algorithm has to deal with all the nodes at each single iteration separately. This causes high running time. The multilevel technique solves this problem by dividing the whole process into several levels. Each level \( i \) has \( k_i \) nodes and it draws these \( k_i \) nodes within a number of iterations by applying a force-directed algorithm. Level \( k_0 \) contains a few coarsest nodes. After each level the number of nodes increases by breaking coarser nodes into the nodes they are formed of, and again some iterations of a force-directed algorithms are applied in order to refine the newly added nodes. This
process continues until $k_i = |N|$. By applying the multilevel technique, the algorithm does not have to deal with $|N|$ nodes from the first iterations which improves the running time. A good survey on different proposed multilevel techniques is the one by Bartel et al. [9].

**Quad Tree**
Quad tree is a technique that approximates the problem of N-body problem. Quad tree reduces the complexity from $O(|N|^2)$ to $O(|N|\log |N|)$ [7].

**Centering Matrix**
The centering matrix $C_n$ of size $|N|$ is defined as:

$$C_n = I_n - \frac{1}{n} \mathbf{1} \quad (2.1)$$

$\mathbf{1}$ is $|N| \times |N|$ square matrix of all 1s.

The centering matrix has application in MDS algorithms (see page 37).

**Power Iteration**
Power iteration is an algorithm that approximates the largest eigenvalue with its corresponding eigenvector of a matrix. It was first introduced by Von Mises in 1929. Power iteration is able to approximate the second largest eigenvalue of the matrix when the largest eigenvalue is found. Power iteration has application in some of the spectral graph drawing algorithms [21, 22].

**Hasse Diagram**
The Hasse diagram is a graphical representation of a locally ordered set. The Hasse diagram is also an upward graph drawing when the dataset is a directed graph [128].

**Scale-Free Networks**
A scale-free network (graph) is a network in which the degree distribution follows a power law. Several different types of graphs exhibit the characteristics of a scale-free networks such as social or biological networks.
Trigonometrical Approximation

Trigonometrical Approximation of \( \sin \) and \( \cos \) or TASC is an approximation approach to calculate the \( \sin \) and \( \cos \) of the angle created by the \( x-axis \) and the line segment between a pair of nodes, (see Section 3.3.4).

INP: Initial Node Placement

All of the iterative graph drawing algorithms start with an initial node placement. For the reason of efficiency most of them start with a random initial node placement. Hu et al. [63] proposed an algorithm for drawing trees with force-directed graph drawing algorithm that places the nodes initially in a way that helps the force-directed algorithm end up with fewer edge crossings compared to a random initial placement.

2.2 A Review of Graph Drawing Algorithms

2.2.1 The Class of Force-Directed Algorithms (Energy-Based Algorithms)

The class of force-directed algorithms are a group of algorithms which exploit information about the connectivity of a graph regardless of the nature of the data. This class of algorithms starts with an initial node placement and then forms the layout over several iterations. Force-directed algorithms define a certain amount of energy for the layout at each iteration. The goal of this class of algorithms is to minimize the energy of the layout over a number of iterations. The definition of energy depends on the strategy of the algorithm. In the following sections, some of these algorithms are discussed.

Spring Embedder

Spring Embedder refers to a group of algorithms which simulate a spring between each pair of nodes. These springs have repulsion and/or attraction forces applied between their two heads. The sum of all repulsion and attraction forces at a particular moment indicates the current energy of the system (layout). Then forces push or pull nodes towards each other in order to minimize the energy and refine the layout.

1. Eades Algorithm

One of the very first algorithms in this category is the one proposed by Eades [29].
This algorithm uses logarithmic strength spring \( c_1 \log \frac{d(n_i, n_j)}{c_2} \) between adjacent nodes where \( c_1 \) and \( c_2 \) are constant and \( d(n_i, n_j) \) is the Euclidean distance between the two adjacent nodes \( n_i \) and \( n_j \) and inverse square force \( \frac{c_3}{\sqrt{d(n_i, n_j)}} \) for non-adjacent nodes \( n_i \) and \( n_j \) where \( c_3 \) is a constant and \( d(n_i, n_j) \) is the Euclidean distance between nodes.

2. Fruchterman and Reingold (FR)

FR \[38\] is one of the well-known algorithms belonging to the class of force-directed algorithms. FR uses simple repulsion and attraction structures between each pair of nodes (e.g. \( n_i \) and \( n_j \)); therefore the implementation is simple:

\[
\begin{align*}
\text{Attraction-Force} &= \frac{d(n_i, n_j)^2}{k} \quad \text{if } \{n_i, n_j\} \in E \\
\text{Repulsion-Force} &= \frac{-k^2}{d(n_i, n_j)} \quad \text{if } n_i \neq n_j; n_i, n_j \in N
\end{align*}
\]

in which \( d(n_i, n_j) \) is the Euclidean distance between \( n_i \) and \( n_j \) and \( k \) is the ideal distance which is calculated as follows:

\[
k = \sqrt{\frac{\text{area}}{|N|}}
\]

FR starts with a random initial placement and moves nodes based on the sum of attraction and repulsion forces at each node. The total energy of the system is the sum of all repulsion and attraction forces (see Equation 2.2).

\[
Energy = \sum_{n_i, n_j \in N \atop i \neq j} \frac{-k^2}{d(n_i, n_j)} + \sum_{\{n_i, n_j\} \in E} \frac{d(n_i, n_j)^2}{k} \tag{2.2}
\]

FR starts with a high nodes’ displacement rate (also described as high temperature, which means it lets nodes have large displacement) and after each iteration the amount of nodes’ displacement decreases. As the nodes’ coordinates are getting more meaningful (i.e., allow for showing more details of the layout clearly) there is less need for large node displacement. The process of controlling the nodes’ displacement rate is similar to the simulated annealing process.

As mentioned earlier, FR starts with a random initial node placement. This random initial node placement has a big effect on the final layout especially for star-shaped graphs. Star-shaped graphs are discussed in Chapter 4. The quality of the final layout by FR depends on the choice of the initial node placement \[116\]. In these cases if the initial node placement is chosen randomly then there is always a chance of having a layout with low quality. A low quality layout can be interpreted as a layout with edge-crossing(s) for planar graphs or avoidable edge-crossing(s) for non-planar graphs. Figure 2.1 shows two layouts for a graph drawn by FR with two different initial node placements.
Although FR is one of the popular force-directed algorithms, it has a few drawbacks. First of all, FR is a very expensive algorithm for large graphs (over one thousand nodes). The time complexity of FR at each iteration is $O(|N|^2)$ as it needs to apply a repulsion force on all pairs of nodes; however, there are a few variations of FR which ignore the repulsion forces for nodes placed very far from each other and this reduces FR’s time complexity for very sparse graphs. Another drawback is that the algorithm may get trapped in a layout whose energy is a local-minimum of the energy function. That locally minimal energy state has usually one or several edge crossings that can be avoided by choosing a different initial node placement.

**Graph Theoretic Distance Algorithms**

Another subclass of force-directed algorithms is the group of algorithms which make use of graph theoretic distance and try to finally accommodate all nodes in a way that the geometric distances between nodes match with graph theoretical distances [78]. This subclass of graph drawing algorithms aims at minimizing the energy of the system by setting up a spring system. The energy of the system in this class of algorithms is called *stress*. Stress in a very general sense is the sum of difference between the geometric distances and graph theoretical distances for all pairs of nodes.

1. Kamada and Kawai (KK)

   KK [70] is a graph drawing algorithm belong to the class of force-directed
2.2 A Review of Graph Drawing Algorithms

KK tries to optimize the energy of the graph layout. This algorithm is sometimes called *graph theoretic distances approach* [78]. Unlike the FR approach which tries to keep adjacent nodes closer to each other and non-adjacent farther, KK makes use of the theoretical distances between nodes in order to find nodes’ coordinates. In other words, a good layout by KK is a layout in which the pairwise geometric distances between nodes match the graph theoretical pairwise distances between nodes. Thus, finding *All-Pairs-Shortest-Path* is a pre-process for KK. The energy of the graph at each iteration by KK is calculated in Equation 2.3 as follows:

\[
\text{Energy} = \sum_{(i,j) \in N \atop i \neq j} \frac{1}{2} k(n_i, n_j)(d(n_i, n_j) - ID(n_i, n_j))^2 
\]  

Here \(d(n_i, n_j)\) is the Euclidean distance between two nodes \(n_i\) and \(n_j\). \(k(n_i, n_j)\) is calculated by \(k(n_i, n_j) = \frac{K}{SP(n_i, n_j)^2}\) in which \(K\) is a constant and \(SP(n_i, n_j)\) is the theoretical distance or shortest path between nodes \(n_i\) and \(n_j\). \(ID(n_i, n_j)\) is the ideal distance between two nodes \(n_i\) and \(n_j\) and it is calculated by \(ID(n_i, n_j) = L \times SP(n_i, n_j)\) and \(L\) is a constant and it is a desirable length for an edge. KK uses the *Newton-Raphson* approach [70] to minimize the energy function in order to estimate the geometric coordinates of nodes.

The KK algorithm draws any undirected graph using the theoretical distances between all pairs of nodes (shortest paths between pairs of nodes). This helps the layout to preserve the structure of the graph in the visual representation. This is because the Euclidean distances between nodes correspond to their theoretical distances.

KK has two drawbacks; first of all the time complexity of KK is high as it needs to calculate *All-Pairs-Shortest-Path* which needs \(O(|N|^3)\) by *Floyd-Warshall* or \(O(|N|^2 \log |N| + |E||N|)\) by *Johnson’s algorithm*. Apart from that, the algorithm needs \(O(|N|^2)\) storage to keep the distances during the whole process of drawing. The algorithm starts with a random initial placement and it initially calculates all the required parameters such as \(k(n_i, n_j), ID(n_i, n_j)\) and \(d(n_i, n_j)\) and then starts refining the nodes’ placement according to their theoretical distances. Another problem of KK is the *local-minima*. Harel et al. [54] state that Kamada-Kawai’s method is trapped in many local-minima which results in some edge crossings. Similar to FR, this happens as a result of initial node placement’s
Background and Literature Review

2. Stress Majorization (SM)

Gansner et al. [39] have proposed an algorithm using the idea of Kamada and Kawai based on graph theoretic distances. Similar to the algorithm proposed by Kamada and Kawai (KK), SM tries to minimize the Stress Function (energy function). The applied method in SM is slightly different from the applied method in KK. As mentioned earlier, KK uses Newton-Raphson method to minimize the energy function which is very likely to be trapped into a local-minimum while SM uses Majorization in order to minimize the stress function. The stress majorization method has shown a number of advantages over the method by Kamada and Kawai.

Using majorization has shown an improvement in running time and stability compared to KK [39]. SM has also shown an improvement in local-minima which results in fewer edge crossings. The stress function in SM is almost the same as the energy function in KK which can be solved either using Cholesky factorization or Conjugate Gradient [46]. However Choesky factorization is recommended for large graphs (over 10,000 nodes).

Figure 2.3 shows two layouts using weighted edges (weighted edges are obtained using the degree of two endpoints of the edge), the layout in Figure 2.3b is drawn...
2.2 A Review of Graph Drawing Algorithms

(a) A layout by Stress Majorization using weighted edges [39].
(b) A layout by Kamada and Kawai using weighted edges [39].

Fig. 2.3 Two different layouts by KK and SM on the same graph.

by KK and the layout in Figure 2.3a is drawn by SM (the layouts are from the work by Gansner et al. [39]).

Binary Stress Model for Graph Drawing

Another proposed algorithm by Koren et al. [82] for graph drawing, bridges the Stress functions and the Spring model in order to draw a graph in a circular drawing area. This algorithm (BStress) spreads nodes within a circular shape which sometimes helps an efficient utilization of drawing area. BStress has two functions to minimize. This algorithm is targeted at minimizing the edge lengths by minimizing the cost function in Equation 2.4.

\[
H(p) = \sum_{\{n_i, n_j\} \in E} (d(n_i, n_j))^2 
\]  

(2.4)

Equation 2.4 tries to minimize the edge lengths which it might also cause node overlapping. Therefore there should be a way to prevent nodes from overlapping. To do that another cost function is defined in this work in order to prevent nodes from collapsing. The second cost function is trying to accommodate all nodes, regardless of their connectivities, with uniform pairwise distances, (see Equation 2.5).

\[
G(p) = \sum_{\{n_i, n_j\} \in N \atop i \neq j} (d(n_i, n_j) - 1)^2 
\]  

(2.5)
Having two stress functions is the reason for choosing the name *Binary Stress* (BStress). Koren *et al.* [82] exploit the majorization optimization technique in order to optimize the sum of two stress functions using the conjugate gradient. The complexity of this approach is $O(|N|+|E|)$ at each iteration. BStress makes use of a parameter $a$ in order to control the edge length while having uniform node spreading. By increasing the value of $a$ the edge lengths get shorter which influences the appearance of the drawing (layout). This parameter helps the algorithm to significantly avoid local-minima.

The final layout by BStress tends to be formed in a circular area. The fixed distance length is decided in Equation 2.5 which is by default set at 1, however 1 can be replaced by any other positive integer value and the result will remain the same.

Figures 2.4a and 2.4b show two drawings for two different big graphs using the BStress algorithm; these layouts are taken from [82]. Figures 2.5a and 2.5b show two smaller graphs using BStress.

![Fig. 2.4 Two graphs by BStress algorithm.](image1)

(a) A layout by BStress with 1933 nodes and 2043 edges.

(b) A layout by BStress with 3487 nodes and 3486 edges.

![Fig. 2.5 Two graphs by BStress algorithm.](image2)

(a) A layout by BStress with 51 nodes and 63 edges.

(b) A layout by BStress with 90 nodes and 89 edges (tree).
2.2 A Review of Graph Drawing Algorithms

2.2.2 Multilevel Paradigm

The class of force-directed algorithms is one of the most successful classes of algorithms in the field of graph visualization. One of the advantages of this class of algorithms over other type of algorithms is that they are easy to implement and they are reasonably fast for moderate size graphs. However this class of algorithms suffers from a few drawbacks. One of the very common problems in this class of algorithm is that they usually converge to a state which locally has minimum energy but it is not a global minimum. This problem happens as a result of random initial node placements. The random initial node placement is the start point of all force-directed graph drawing algorithms. Another drawback which this class of algorithms suffers is the running time for large graphs. Although force-directed algorithms are relatively fast, since they usually have cubic complexity (including the number of iterations) they can be very slow for large graphs. Walshaw [122] introduce a new approach, the Multilevel Paradigm, which can be incorporated with the class of force-directed algorithms and can significantly help to overcome on the aforementioned drawbacks.

The most tangible difference between the class of multilevel algorithms and the traditional force-directed algorithms is that multilevel algorithms do not introduce all of the nodes to the drawing from the start point of the algorithm. Instead, nodes are gradually introduced over a number of levels. Nodes in multilevel algorithm are initially represented as a big coarsened node(s). At the beginning of each level, the current coarsened node(s) are broken down into smaller ones. This process continues until all coarsened nodes become single nodes. Multilevel algorithms start with the first level which contains one or a few coarsened node(s) and refines the positions of those current coarsened nodes with a force-directed algorithm. Then, in the next level the coarsened nodes are broken down and the drawing gets refined and so on.

Bartel et al. [9] have divided the multilevel paradigm into three main phases which are common for almost all multilevel algorithms:

1. Coarsening
2. Placement
3. Local layout or Single level layout

1. Coarsening

As mentioned earlier, the multilevel algorithms do not start with all the single nodes but one or a few coarsened nodes or super-node(s). A super-node is
Background and Literature Review

composed of a few smaller super-nodes or single-nodes which are collapsed in some fashion. There are several different approaches to convert all the nodes into one or a few coarsened nodes. One of the proposed approaches in the literature is Matching Merger or Edge Collapsing. This approach creates the coarsened graph level by level. At the very first level $G_0$, pairs of adjacent nodes are visited and each pair is collapsed into a coarser node. If any node remains alone that node becomes a coarser node on its own. A weight is assigned to each coarser node which is usually the number of single nodes that creates the coarser node. The same process happens in the next level $G_1$ where all nodes (coarsened nodes from previous level) become unvisited. This process (coarsening) continues until the whole graph collapses to only 1 or a few coarsest nodes. Figure 2.6 shows a simple example of a graph coarsening approach.

![Fig. 2.6 A graph with 6 nodes being coarsened using a simple Matching Merger approach.](image)

A few other coarsening methods exist in the literature such as Random Merger [9] which starts by selecting random nodes and matching them to a random neighbor. Weighted Matching Merger is a modified version of Matching Merger in which each node selects the unvisited neighbor with smallest weight. Maximum Independent Vertex Set (MVIS) is another coarsening method in the literature. MVIS creates sub-sets of nodes in which no two nodes in one sub-set are adjacent [9]. Hadany and Harel [52] have aimed at creating the coarser nodes starting with nodes with smaller degrees. Hu [60] uses two different methods for coarsening: Edge Collapsing and MVIS. Walshaw [122] also uses a method similar to Edge Collapsing.

The idea of node coarsening is inspired by the field of graph partitioning [56, 123] in which nodes of a graph fall into groups of nodes based on similarity in terms
2.2 A Review of Graph Drawing Algorithms

of their connectivity. This idea practically helps avoiding local-minima of the energy function.

2. Placement

At each level a multilevel algorithm refines the positions of the newly added nodes from the process of un-coarsening, i.e. nodes added after the previous level. To refine the positions of nodes, first of all the new un-coarsened nodes have to be somehow introduced and initiated to the layout. The process of initiating new un-coarsened nodes is called Placement.

Normally new un-coarsened nodes should be accommodated relatively close to their parent (super-node). This is because those new un-coarsened nodes belong to a specific super-node and have strong relations with each other and their parent (super-node). There are different methods for Placement in the literature. A smart placement can improve the quality and even running time of the algorithm. A few different placement methods have been introduced in the literature so far [9]. Typically, if nodes $n_i, n_j \in G_i$ are derived from a coarser node $w \in G_{i+1}$ then the positions of $n_i$ and $n_j$ should be very close to $w$. Some of the algorithms place the new nodes within or on a circle centered at $w$ with a very small radius. Figure 2.7 shows an example of a placement method.

3. Local layout

To prepare the layout in level $G_{i+1}$ a single level graph drawing algorithm is needed in order to take the new introduced un-coarsened nodes and refine their position to progress the system to level $G_i$. Single level graph drawing algorithms start with $|N|$ nodes and compute the layout of those $|N|$ nodes at the end. Usually, multilevel algorithms make use of a traditional force-directed algorithm such as Fruchterman and Reingold (FR). Hu [60] and Walshaw [122] make use of an adapted version of (FR) in their algorithm, in order to compute the layout during each level, while Harel and Koren [54] make use of an adapted version of Kamada-Kawai (KK).

The most significant difference between multilevel algorithms and single force-directed algorithms is the improvement of running time. Multilevel algorithms also improve the chance of avoiding local-minima [60], however there is no guarantee of reaching global minimum. For instance, Figure 2.8 shows a layout of a large graph called jagmesh7 by Hu’s algorithm implemented in Graphviz [41]. As illustrated, the layout is folded and created a local-minimum. This folding creates many edge-crossings as the graph is large.
3.2.3 Graph Drawing using Evolutionary Algorithms

Genetic Algorithm in Graph Drawing

The genetic algorithm (GA) is a well-known method for heuristically solving combinatorial optimization problems with techniques inspired by natural evolution such as selection, crossover and mutation. Genetic mutation is an example of a phenomenon in which a set of genes within a chromosome are altered to produce a chromosome
more fit to the task in hand. The genetic algorithm is a subset of the more general class of evolutionary algorithms (EAs) [32]. GAs have applications in many different disciplines such as bioinformatics, engineering, chemistry, economics, mathematics and also in geometry. The GA is mostly applied on types of problems that involve a solution (inspired by chromosome) in the form of bits (inspired by genes) which can have different types of values such as numerical, binary or even characters. The GA normally starts with a certain number of random orderings (chromosomes or solutions) and calls them a population. The top solutions from the population are selected based on a cost function or so called fitness function for modification with the hope of getting improved for the next generation. The typical genetic algorithm has a few techniques in order to modify a given population of solutions. Selection, Cross Over and Mutation are the three common GA techniques for modifying the solutions. Larranaga et al. [83] show different versions of these techniques using the well known Traveling Salesman problem.

Within the field of graph drawing GA has several applications. The most important phase in using GA in the field of graph drawing is to choose an appropriate fitness function. Normally the parameters of the fitness function correspond with the layout aesthetic criteria. For instance, graph drawing algorithms may try to minimize the number of edge-crossings, node cluttering and also the edge lengths. Hence, the fitness function has to be designed in a way to take care of these aesthetic criteria. Branke et al. [16] proposed a GA for graph drawing which uses a force-directed algorithm for improving the layouts. They applied a genetic algorithm combined with a force directed method as a local fine-tuner within the genetic algorithm, such that after each generation the force-directed algorithm is applied on the resulting layout until next iteration of the force directed method would no longer improve the quality of the graph. This fine tuning helps to improve the quality of the final layout at the end but the running time is increased. The fitness function in their GA is based on a few parameters such as the number of edge crossings, the variance of edge lengths, the minimal distance of a node to an edge, and the average resulting spring-forces in the improved force-directed layouts. Barreto et al. [8] proposed another GA for straight-line drawing of undirected graphs. They explored several aesthetic criteria for the fitness function, including the value of the energy function of the force-directed algorithm of Kamada and Kawai which minimizes the dissimilarity of the Euclidean distance between each pair of nodes and the graph theoretical distance between them. They report that this criterion alone leads to uniform edge lengths and uniform vertex distribution but it does not always result in an overall aesthetically acceptable solution.
Background and Literature Review

Some other aesthetic criteria that they explored include the number of edge crossings, edge length deviation and distance between nodes and edges. Two different crossover operators, absolute crossover (AX) and relative crossover (RX) (see [83] for details), are applied sequentially in this GA. Figure 2.9 shows 4 different layouts for a highly symmetrical graph with 19 nodes resulting from different number of generations.

Another work on applying a GA for drawing undirected graphs is the TimGA algorithm proposed by Eloranta and Makinen [33]. Similar to the other GAs described above, TimGA utilizes a fitness function based on aesthetic criteria such as the number of edge crossings, even distribution of nodes and edge length deviation. The layout of a graph $G = (N, E)$ is represented as an $|N| \times |N|$ matrix, i.e. unlike the GAs described above, the drawing area in TimGA is discrete with nodes being assigned to the cells of a matrix. TimGA uses elitist selection which means that the best chromosomes (here representing layouts) are always selected for the next generation. Two crossover operators are used: *RectCrossover* and *ThreeNodeCrossover*. The most elaborate genetic operator in TimGA is its mutation which actually consists of 16 different mutation operators such as *SingleMutate* and *SmallMutate*, as introduced by Groves et al. [51] in their work on applying GA for graph drawing. SingleMutate chooses a random vertex and moves it to a random empty cell, while SmallMutate randomly chooses two cells, at least one of which is occupied, and swaps them. One of the drawbacks of TimGA is that it cannot properly draw some simple graphs, e.g. cycle graphs. Figure 2.10 shows a few layouts by TimGA.

Genetic algorithms are flexible methods in the field of graph drawing as any new criterion with different level of importance can be embedded easily in the fitness function. The most important drawback in using genetic algorithm in the field of graph drawing is the cost of running time especially when the graph is large.
Simulated Annealing in Graph Drawing

Simulated annealing is a heuristic technique that mimics the process undergone by misplaced atoms in metal when heated up and then slowly cooled down [127]. Kirkpatrick et al. [77] in 1982 introduced the idea of making use of simulated annealing for optimization problems. The idea is to search for possible solutions in order to converge to an optimal solution. The trick in simulated annealing is to start with a large amount of random movements and then decrease the movements gradually [101]. The rate of the movements is proportional to a parameter— the so called temperature of the system. Mutation in most simulated annealing is the same as movements; meaning that the process starts with a high number of mutations (high temperature) and gradually decreases the number of mutations after each iteration until the system reaches the minimum temperature or gets cooled.

Almost all problems that can make use of genetic algorithms can make use of simulated annealing. Davidson et al. [27] make use of simulated annealing in the field of graph drawing in order to nicely draw a graph. The fitness function in their work comprises several different criteria, some of which are listed in below:

1. Node distribution: Spreading nodes evenly.
2. Borderline: Forcing nodes to stay in a limited area.
3. Edge lengths: Uniform edge lengths on average.
4. Minimizing the number of edge crossings.
5. Node-edge distances: Avoiding close distance between nodes and edges.
Similar to the GA, the simulated annealing algorithm proposed by Davidson et al. [27] is also unable to properly draw cycle graphs (see Figure 2.11).

![Cycle graph drawn by Davidson et al. [27].](image)

Fig. 2.11 A cycle graph drawn by Davidson et al. [27].

Brank shows that the applied criteria in Davidson’s work can be achieved using partial differentiation and minimization using gradient descent [15]. In this work the positive and negative points of both methods were discussed.

### 2.2.4 Spectral Matrix Graph Drawing Algorithms

Graphs, or any other type of data with available pairwise relational information, can be represented by a square matrix. A graph can be represented by an adjacency matrix, edge list or any known pairwise relational information that can be represented by similarity/dissimilarity matrix or sometimes a distance matrix. There is a class of data visualization techniques which make use of the matrix representation of the data in non-iterative schema by decomposing the matrix. Hall [75] proposed the very first spectral graph drawing algorithm in 1970. Unlike the class of force-directed algorithms, this class of algorithms uses the spectral information of the matrix representation. The spectral information of a matrix representation comprises of eigenvalue, eigenvector and Laplacian matrix and they are used in order to find the Cartesian coordinates of
data individuals or graph nodes. There are several works in this area that use spectral information to draw undirected graphs [80, 22, 21]. In the following sections a few spectral matrix graph drawing algorithms are discussed.

**Multidimensional Scaling (MDS)**

MDS is a generic name for a group of visualization methods that visualize data using a distance matrix. Classical MDS, Metric MDS and Generalized MDS are some of the methods in this group. Classical MDS (CMDS) [67], for example, takes the distance matrix of the data and calculates the eigenvalue decomposition of the matrix in order to find the coordinates of nodes in the graph. CMDS finds the 2 or 3 (if 3D) largest eigenvalues of a specific matrix $M$ created by the distance matrix of the graph by using double matrix centering. Then corresponding eigenvectors of those 2/3 largest eigenvalues are considered as the 2/3 vector dimensions of the data.

**Spectral Distance Embedding Graph Drawing**

Spectral distance embedding graph drawing algorithms are algorithms that make use of a distance matrix in order to find the coordinates of the nodes.

Two algorithms belong to this class are:

1. SDE: Graph Drawing Using Spectral Distance Embedding.
   SDE is an algorithm that draws undirected graphs using their distance matrix [21]. SDE has two phases:
   
   (a) Computing all-pairs-shortest-path. Civril et al. [21] proposed a BFS algorithm in order to compute the distance matrix.

   (b) Finding the largest eigenvalues of a specific matrix called $M$ (double matrix centering). SDE applies power-iteration algorithm in order to find the largest eigenvalues.

   The running time to draw a graph with around 10000 nodes is around 45 seconds in which around 28 seconds is for computing the distance matrix and 17 seconds is for the power-iteration phase, using Pentium 3 PC.

2. SSDE: Fast Graph Drawing Using Sampled Spectral Distance Embedding.
   SSDE is another algorithm from the same family as SDE. It has an improvement in a sense that it does not compute all-pairs-shortest-path (an accurate computation of distance matrix) but an approximation of distance matrix by sampling $C$
nodes instead of $|N|$ nodes where $C$ is a very small number compared with $|N|$. SSDE is not an appropriate method to visualize sparse graphs with low density and also it is not suitable for very dense graphs as the sampled nodes are not able to return enough information about the distance matrix.

One of the disadvantages of this class of graph drawing algorithm is that in the case of real-time visualization when a new node is added the whole graph has to be re-drawn again as it does not use iterative schema to keep the current positions. Sometimes when a new node is added only a minimum refinement is required in order to refine the position of the graph with the new added node.

### 2.2.5 Hierarchical Graph Drawing Algorithms

Most of the graph drawing algorithms that have been referred to in the previous sections are usually generic algorithms in a sense that they can be applied on an undirected with almost any type of topology. There are other types of graphs such as directed graph or digraph. There are specific graph drawing algorithms for drawing digraphs known as layered graph drawing algorithms. Layered graph drawing algorithms use several layers either horizontal or vertical where nodes can be accommodated. Healy et al. [55, 109] proposed an algorithm that visualizes directed graphs with unidirectional edges which can incorporate some information about node and edge widths. Hong et al. [59] proposed an algorithm for drawing directed graphs in 3D space which is an extension of the Sugiyama [108] method for 2D layered drawing.

### Radial Graph Drawing Algorithms

Another approach to layered graph drawing is using radial or circular layers. One of the very first attempts of radial drawing was introduced by Reginani [99] in which a Ring Diagram (RD) layout for some (not all) hierarchical graphs is presented as a replacement for HD (Hasse Diagram) that tries to reduce the number of edge crossings compared to HD. Giacomo et al. [45] proposed a radial graph drawing with a minimum number of concentric circle for planar graphs. Bachmaier [3] introduced another radial graph drawing algorithm which is an adaptation of the Sugiyama method and instead of using horizontal lines, nodes are placed on the border of some radial circles. This means that edges are placed along the border of a circle and have two possible directions. This choice reports a reduction of edge crossings by about 30%.
2.2.6 Other Graph Drawing Algorithms

One of the most important aesthetic criteria in a graph layout is to have a low (minimum) number of edge-crossings. This is one of the most common concerns for most of the graph drawing algorithms. Running time is another concern as large graphs may need long running time to be rendered. Although these aforementioned criteria are important, there are other criteria which might be important for some specific graphs. Most of the graph drawing algorithms try to produce a layout with approximately equal lengths for edges. Sometimes it is better to have longer lengths for some specific edges and shorter for others. A few algorithms with specific features (e.g. using pre-defined edge-lengths, circular drawing) are:

1. **Maxent-Stress**: Gansner *et al.* [40] introduced an algorithm in which a new term (the *Entropy of a layout*) is added into the stress function. The objective of this algorithm is to satisfy pre-defined edge lengths. This algorithm aims at optimizing the maxent-stress function, while it does not need to calculate all-pairs-shortest-path. Another work with the same objective (edge length-sensitive) is presented by Ganser *et al.* [42].

2. **Sparse Stress Model**: Ortmann *et al.* [89] introduced a new algorithm which reduces the complexity of the traditional force-directed approach or stress function models from $O(|N|^2)$ to $O(k|N|+|E|)$ where $G = (N, E)$ and $k$ is a parameter called the pivot and its value is proportional to the value of $|N|$. This algorithm needs a pre-processing step with time complexity $O(k(|E|+|N| log |N|))$.

3. **Crossing Reduction in Circular Layouts**: Baur *et al.* [13] proposed a model for drawing graphs on the border of a circle with a minimum number of edge crossings. Their model consists of two phases, the first phase creates an initial circular layout by making use of a greedy approach and the second phase further optimizes the result of the first phase. The second phase initially calculates the total number of edge crossings of the layout.

2.2.7 3D Graph Drawing

3D visualization has applications in a few different fields such as game and design, medical device simulation and etc. 3D graph drawing can be utilized on a sphere to present data that have an inherent 3D nature. There are several different 3D graph drawing methods in the literature. Some of them are listed below:
1. Ostry [90] who discussed the issues of extending 2D force-directed graph drawing algorithm to 3D graph drawing.

2. Gotsman et al. [48] investigate on the problem of mesh parameterization and its usage for spherical parameterization of 3D meshes.

3. Kobourov et al. [79] show how the traditional force directed algorithms using two forces (attraction and repulsion) can be applied in Riemannian space with spherical and hyperbolic geometry. In the case of spherical geometry any force directed algorithm can be extended to 3D by mapping points to and from tangent space so that points always stay on the surface of a sphere.

4. Munzner [88] propose a 3D graph drawing in hyperbolic space using a spanning tree instead of the whole graph.

2.3 High Dimensional Data Visualization

A dataset with a large number of features is called high dimensional data. A dimension here refers to a feature, therefore, a data individual with a larger number of features is a datum with high dimensionality. For instance a text document is considered a high dimensional data individual with the words occurring in the document considered as its features. Individuals with high number of similar features (dimensions) are said to be more similar. Projecting (visualizing) high dimensional data into lower dimensional space (2D or 3D) gives a better general view of the data. Therefore, one can use the similarity/dissimilarity between all pairs of data individuals in order to visualize the data. As an example, by visualizing the text documents from different disciplines using their similarity/dissimilarity matrix one can distinguish between text documents from different disciplines if the visualization technique is accurate enough [85]. There are some techniques that make use of the force-directed approach in order to visualize high dimensional data such as the work proposed by Lespinate et al. [84].

In this section, we discuss the major high dimensional data visualization or dimensionality reduction techniques:

2.3.1 Multidimensional Scaling

The class of multidimensional data projection techniques is another visualization approach which deals with high dimensional data when an individual datum has multiple dimensions. This approach needs the similarity/dissimilarity of the data studied in
order to approximate the coordinates of the individuals in a lower dimensional space (2D or 3D). Figure 2.12 shows an example of one of the most popular multidimensional data projection technique called classical multidimensional scaling [67]. This approach is a subset of the large class of multidimensional scaling techniques [26].

![Fig. 2.12 An example of a classical MDS [67] for a data with 2000 data individuals with 5 different groups in which there is no similarity between any two groups; groups are colored with different colors.](image)

### 2.3.2 Sammon Projection

Sammon’s projection [105] is a visualization technique that takes a similarity/dissimilarity matrix of the data and visualizes it in 2D or 3D space. Sammon’s projection is similar to MDS but with less emphasis on pairs with higher similarities. Figure 2.13 shows the visualization by Sammon’s projection for the same data visualized in Figure 2.12.

### 2.3.3 k-nearest Neighbors Techniques for High Dimensional Data Visualization

Some data visualization techniques make use of the k-nearest neighbors [111] in order to lay out the data in a lower dimensional space.
Fig. 2.13 An example of a Sammon projection for data-set with 2000 data individuals with 5 different groups in which there is no similarity between any two groups; groups are coloured with different colours.

**LargeVis**

LargeVis is a visualization technique that approximates the k-nearest neighbor of each node according to the similarity matrix in a linear time complexity [110].

**t-SNE**

Maaten et al. [85] introduce a technique called t-Distributed Stochastic Neighbor Embedding (t-SNE) in order to visualize high dimensional data [85]. The t-SNE is a variation of the Stochastic Neighbor Embedding technique that uses the nearest neighbor to project and approximate the coordinates of the data individual. t-SNE is the state-of-the-art technique for visualizing high-dimensional data.¹

¹t-SNE has been mentioned as the state-of-the-art in the field of data visualization several time in the recent few years by some conference reviewers.
Chapter 3

Concentric Approach in Graph Drawing using Distance Matrix

3.1 Introduction

We propose a new approach, called the concentric approach, for information visualization when pairwise information between all pairs of individuals in a dataset is available. The pairwise information is a numeric expression of the relation between each pair of individuals. It can be represented as a binary value (0 or 1), if the data is a graph; i.e. each pair of nodes is either adjacent so that their pairwise information is 1, or is non-adjacent so the pairwise information is 0 [14, 129]. The pairwise information between pairs of individuals can also be represented as a non-discrete value or a percentage of similarity/dissimilarity of the individuals, especially when dealing with multidimensional data [66, 17].

This approach can deal with the two aforementioned different types of data above (graphs and multidimensional data). In the field of graph drawing, the pairwise information between all nodes is represented in the form of an adjacency matrix, an edge list, an incidence matrix or a theoretical distance matrix.

One of the characteristics of the concentric approach is to place the center of gravity of the drawing almost at the origin of a $D$-dimensional drawing space. This feature can be interesting for some applications which are interested in finding the gravity of the convex hull of a layout. Figure 3.1 illustrates an example of a drawing at the center of a 2D space. The sum of the coordinates of all nodes in this type of drawing is almost zero.
Similar to force-directed graph drawing algorithms, the concentric approach is an iterative approach. It always starts with an initial node positioning (random positioning or smart positioning) and draws the graph over a number of iterations.

As mentioned earlier, our proposed algorithms in this chapter and the next chapter do not calculate any energy or stress of a layout, and unlike the other force-directed algorithms, our algorithms do not resemble any mechanical system. Therefore, the termination of our algorithm is based on the predefined number of iterations that depends on the size of a given graph. This will be discussed later in this chapter. Although the term local-minimum is used when a parameter such as stress or energy is being minimized and trapped into a locally minimum state, we make use of this term to indicate visual cluttering such as the presence of avoidable edge crossings and layout folding.

### 3.2 Concentric Approach

In this section the general view of the concentric approach and its use in data visualization is presented.

#### 3.2.1 General View

Concentric is a term that we use in order to describe our proposed approach. This approach is applied to the proposed algorithms in this chapter and Chapter 4. The concentric approach refers to multiple circles centered at the origin of the Cartesian
system. Unlike the other concentric approaches in the field of graph drawing [45, 13], our approach does not necessarily produce circular-shape layouts, however sometimes the layouts can be circular as can be seen in Chapter 4 Section 4.2.1. Our strategy is completely different from the previously developed algorithms for circular shape graph drawing [45, 13, 82, 4].

### 3.2.2 Concentric Approach in Visualization

In this chapter we introduce two different algorithms based on the concentric approach. These two algorithms have different applications:

1. **CTD**: Concentric Approach in Graph Drawing using Theoretical Distance

2. **CMD**: Concentric Approach in Multidimensional Data Visualization using Dissimilarity Matrix

CTD is an algorithm in the field of graph drawing, and the input data for this algorithm is the distance matrix of a graph (see Section 2.1). CMD is an algorithm to visualize multidimensional data when the similarity/dissimilarity matrix of a multidimensional data is given (see Section 2.1).

The input data of the CTD and CMD algorithms are similar to each other in a sense that both of them represent the differences between pairs of individuals and they are different because the distance matrix is calculated from the adjacency matrix or edge list of a given graph, so that the shortest distance between a pair of nodes is the minimum number of edges between them. Therefore, the distance matrix contains discrete values. However, the similarity/dissimilarity matrix has non-discrete or continuous values, and this is because the similarity/dissimilarity matrix data is based on the similarities/dissimilarities of different attributes of each pair of individuals (see Section 2.1). Note: We assume that there are no edges between individuals when multidimensional data is applied.

Therefore, a dataset $N$ with $|N|$ individuals for either CTD or CMD is a symmetrical $|N| \times |N|$ matrix in which the diagonal line is all 0s. In the remainder of this section we give a general example on how the concentric approach is applied on a graph using its distance matrix. In a similar way the concentric approach is applied on a similarity/dissimilarity matrix.

For a graph, $G = (N, E)$ the number of concentric circles is $\frac{|N| \times (|N| - 1)}{2}$, where each circles’ diameter represents the distance between a pair of nodes. Initially, every pair of nodes is placed on the border of a circle with the two nodes being opposite to each
other and the diameter of the circle being equivalent to the theoretical distance between
the two nodes from their distance matrix. Since there are $\frac{|N|\times(|N|-1)}{2}$ distinct pairs of
nodes, there are $\frac{|N|\times(|N|-1)}{2}$ circles as well. Note that nodes have multiple copies at this
stage. Figure 3.2 shows how concentric circles are formed based on the given distance
matrix.

Fig. 3.2 An example of a theoretical distance matrix for a graph with four nodes and
its corresponding concentric circles according to the concentric approach.

Nodes at the very first step are distributed somewhere on their corresponding circles.
For a graph $G = (N, E)$ with $|N|$ nodes there are $|N| - 1$ copies of each node. Figure 3.3
shows an example of the node arrangement in a concentric layout for a graph with 4
nodes. For example, the triangle node is associated with three different distances (2, 1,
6) so there is one copy of triangle on the border of the orange circle with radius 2, one
copy on the red circle with radius 1 and one copy on the brown circle with radius 6.

As illustrated in Figure 3.3, each node has $|N| - 1$ different copies. Next, the
positions of nodes have to be refined. Figure 3.4 is an example of a small graph with
3 nodes with their distance matrix illustrated in Table 3.1. Since there are 3 nodes,
3.2 Concentric Approach

Fig. 3.3 An example of the theoretical distance matrix of a graph with four nodes and its corresponding initial layout at the beginning of an iteration of the concentric approach.

there are two copies of each node. We suppose that Figure 3.4a is the first iteration. Nodes are distributed somewhere (randomly if it is the very first iteration) on their corresponding circles according to the distances at Table 3.1. Each iteration can be simplified within two steps when the different copies of nodes are accommodated around the concentric circles:

1. Finding the new updated position for each node.

2. Creating the new set of nodes’ copies on concentric circles for the next iteration.

Finding the new position for each node out of $|N| - 1$ copies is performed by finding the average of the nodes’ positions. Minor modifications might be applied depending on the type of the graph, e.g. using a weighted average. Figure 3.4b shows how two copies of each node are averaged in order to update the position of the corresponding node. For example, the red solid colored node is the result of the averaging of two
Table 3.1 Distance matrix for a graph with 3 nodes.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Green</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Blue</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

red non-solid colored nodes. Figure 3.4c shows the updated positions of nodes at iteration $t$, however, they are not necessarily on the borders of concentric circles. In order to move on to iteration $t + 1$, one needs to create new copies of each new node based on their current positions. The first thing in order to find the new copies of nodes based on the current positions is to consider a straight line between each pair of nodes, (see Figure 3.4d). Each line has to move with exactly the same slope towards the origin of the Cartesian space so that it passes the origin of the Cartesian space, and it has to be re-sized according to the theoretical distance between the two nodes at the two endpoints of the line. Figure 3.4e shows the new modified lines. As it is shown in Figure 3.4e all lines keep their initial slopes from Figure 3.4d and all of them pass the origin of the Cartesian space. Now there are new copies for each node, (see Figure 3.4e).

Moving the lines between the updated nodes in order to create new copies of nodes (moving from Figure 3.4d to Figure 3.4e) is equivalent to the calculation of sine and cosine of the angle between each line and the x-axis and multiplying it by the corresponding theoretical distance between the nodes at the two endpoints of the line. Now Figure 3.4f is equivalent to the Figure 3.4a so the same process can be repeated in order to perform a new iteration.

Figure 3.4 shows the process of updating the positions of nodes out of multiple nodes’ copies in order to move on to the next iteration. To simplify the presentation of our technique and avoid a crowded layout in Figure 3.4, we have used a few nodes (3) in which the average position of a new node is typically the middle of a line from one node to the other, (see Figure 3.4b). For bigger graphs, the new position of a node is calculated by averaging of all node-copies’ positions. The concentric circles of a graph with 4 nodes whose distance matrix is represented in Table 3.2 is illustrated in Figure 3.5a. Figure 3.5b shows how the new position for the green and yellow nodes are calculated by averaging the previous positions of the copies of yellow and green nodes on concentric circles. The green node (node number 1) and the yellow node (node number 2) now represent the updated positions of the yellow and green nodes.
3.2 Concentric Approach

(a) Iteration $t$ with 3 nodes (red, green and blue) on a set of concentric circles using the distance matrix in the Table 3.1.

(b) Averaging of the copies of each node in order to update the position of each node; e.g. the red-colored solid node is the average of two red-colored non-solid nodes.

(c) New updated nodes from Figure 3.4b are accommodated at their new positions in the drawing space.

(d) The lines between each pair of nodes are identified (dotted lines).

(e) The lines from Figure 3.4d are moved and re-sized so that they pass the origin of the Cartesian space and re-sized according to the theoretical distance between the nodes at the two endpoints of each line from Figure 3.4d.

(f) The copies of nodes are accommodated on the concentric circles and ready to update the new positions of nodes at iteration $t + 1$.

Fig. 3.4 Transition from one iteration to the next one.
Table 3.2 Theoretical distance matrix for a graph with 4 nodes.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Red</th>
<th>Yellow</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Yellow</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Blue</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Green</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

The whole process is repeated until the updated positions of nodes are satisfactory. Note: The new updated nodes are not necessarily on the border of any circle.

There are a few different ways to exploit the pairwise distances. One way is to use the raw theoretical distances as they are (a set of positive integer distances). Another option is to raise the theoretical distances to a positive power in order to get a better result as discussed below.

Local-minima are an important concern in the field of graph drawing when minimizing the number of edge-crossings [19, 94, 95]. We have noticed that if the theoretical distances are raised to the power of \( c \) where \( c \) is either 2 or 3 then the chance of getting trapped into a local-minimum is significantly reduced. By doing so the differences between the theoretically shorter distances and the theoretically longer distances become bigger. Our experiment shows that by doing this we can reduce the chance of getting trapped into a local-minimum (see Figure 3.27). Equation 3.1 shows how the theoretical distances are powered. For instance for a graph with 4 nodes with the following theoretical distances (shortest paths): \( SP(1,2) = 1, SP(1,3) = 2, SP(1,4) = 3, SP(2,3) = 1, SP(2,4) = 2, SP(3,4) = 1 \); we can have the raised distances as follow: \( SP(1,2) = 1, SP(1,3) = 4, SP(1,4) = 9, SP(2,3) = 1, SP(2,4) = 4, SP(3,4) = 1 \) so that the distance between nodes 1 and 4 which is originally 3, becomes 9. In other words, one could exaggerate the pairwise nodes’ distances for those pairs with larger theoretical distances by \( c \) where \( c \) is a small positive integer value. By doing so pairs with shorter theoretical distances are brought even closer to each other in order to avoid local-minima. This strategy has to be applied at some initial iterations of the algorithm and then the original theoretical distances can be used. Once the layout is stabilized, after a number of iterations, then the real distances can be used in order to refine the layout.

\[
\text{Raise}.SP(n_i,n_j) = SP(n_i,n_j)^c
\] (3.1)
3.2 Concentric Approach

(a) 6 concentric circles for a graph with 4 nodes whose distance matrix is shown in Table 3.2.

(b) Two examples of finding the average of different copies of the yellow and green nodes on concentric circles in order to update the position of the yellow and green nodes. Node number 1 with the green background represents the updated position for the green nodes and node number 2 with the yellow background represents the updated position for the yellow nodes.

Fig. 3.5 Nodes on different circles are averaged in order to be updated.
The outcome of most of the iterative graph drawing algorithms is influenced to a great degree by the initial node placement (INP) [38, 70]. Most of the currently developed force-directed algorithms use a random INP for reasons of efficiency so that the output sometimes is influenced by them. In Chapter 5, a new pre-processing step applicable to any iterative and single-level force-directed algorithm is proposed. It uses smart INP instead of random INP. Relatively small size graphs can benefit from a smart INP, (see Section 2.1). There are a few works in this area [116–118] which show a significant improvement in quality for relatively small graphs when iterative algorithms are used. These smart initial node placement methods have the same effect when applied in the context of the concentric approach, (see Figure 3.26).

Our proposed approach is an iterative scheme algorithm, therefore, it can adopt the multilevel paradigm and benefits from it when the size of the graph is large. The problem of local-minima which is mainly caused by random initial node placement for large graphs is solved by embedding the multilevel technique into the concentric approach. We propose a new multilevel technique in which the coarsening step of the traditional multilevel algorithm is ignored and nodes are introduced to the layout gradually starting from either a selected node or a random node. Our preliminary results using this multilevel technique are published in [115]. The adopted multilevel technique in the concentric approach does not need a coarsening step which results in reduced time complexity and consequently faster running time. The general view of the concentric approach is illustrated in Figure 3.6.

3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

3.3.1 Introduction

In this section, we present a new algorithm, CTD, for drawing any undirected connected graph. CTD is based on the concentric approach where the matrix to be used is the theoretical distance matrix of a graph. The pioneering work to use theoretical distance for graph drawing is the one by Kamada and Kawai [70], where the placement of the nodes depends on the theoretical distance between them. More detail about previous work in this field is given in Chapter 2.

The bottleneck of this approach, or generally all other approaches that make use of the theoretical distance matrix, is the calculation of this matrix from the binary adjacency matrix of a graph. There are several works in this area such as the well-known
Dijkstra’s algorithm which is extensible for solving the problem of all-pairs-shortest-path (i.e. the theoretical distance matrix). Other works include the Floyd–Warshall algorithm [35] as well as Johnson’s algorithm [68] which are particularly suitable for sparse graphs. A couple of recent works in this field improve the time complexity of the previously developed algorithms [113, 130]. Our focus in this work is not on the all-pairs-shortest-path algorithms, so we are not discussing the efficiency of the all-pairs-shortest-path algorithm, therefore, the reported running time of CTD and any algorithm we compare it to does not include the time spent to calculate the
Concentric Approach in Graph Drawing using Distance Matrix

all-pairs-shortest-path. In this case, one can have a better insight to compare the functionality of the two pure algorithms, (see Table 3.4).

3.3.2 Multilevel Adoption

Traditional force-directed algorithms typically start with all nodes on the screen and refine their positions over a certain number of iterations [38, 70]. Applying the traditional force-directed algorithms to large graphs increases the chance of the layout being trapped at a local-minimum and also increases the running time [19, 78, 9]. The multilevel technique for the iterative (force-directed) algorithms helps avoiding local-minima and reduces the running time [60, 122]. The idea of the multilevel paradigm is to not start the drawing with all nodes but with a few selected ones or with coarsened nodes (see Section 2.1) and then to apply a force-directed algorithm to refine the positions of those nodes. This is considered as the first level. Then new nodes are introduced to the drawing and their positions are refined, which leads to the next level. This process continues until all of the nodes are introduced to the drawing within a number of levels.

The utilization of the multilevel paradigm for force-directed graph drawing has shown a significant improvement (compared to the single-level force-directed graph drawings) in terms of running time and escaping from local-minima [60, 122]. A complete survey of different multilevel approaches is proposed by Bartel et al. [9]. As discussed in Chapter 2, one of the main three phases of the multilevel paradigm is the coarsening step in which nodes are hierarchically coarsened in multiple levels/steps until either one or a few coarsest nodes are left. Then the visualization is executed at multiple levels as well. The visualization starts with the coarsest node(s) and at each level coarsened nodes are broken down. This process continues until nodes cannot be broken down further.

CTD utilizes the described multilevel paradigm above but with a novel approach to the coarsening problem. We call this the **Multilevel Vertex Neighboring** approach (MVN) [115]. MVN can be viewed as an instant coarsening that does not need to be computed in advance in order to create a coarsest(s) node(s). MVN initially marks all the nodes as unvisited nodes. It then starts the force-directed drawing with a few nodes, marks them as visited and draws them using an adapted drawing algorithm. Then after each level, the neighbors of the current nodes are added and become visited nodes. This process continues until all the nodes are visited. This approach, the Multilevel Vertex Neighboring (MVN), was first employed in our work to draw large
graphs but without calculating the distance matrix [115]. The question here is from which node(s) should we start the neighboring method?

We propose two slightly different approaches:

1. Selected nodes to start:
   
   We notice that starting from the corners of a graph reduces the chance of getting trapped into a local-minimum. We define the corners of a graph by having the distance matrix of a graph. A pair of nodes is said to be the two corners of a graph when they have the largest theoretical distance (diameter). We found that if the MVN algorithm starts with the three corners, the drawing can be more efficient. To have the three corners of a graph one can start the drawing from these different corners of the graph until all nodes are visited. To find the three corners of a graph one needs to first find the longest theoretical distance from the distance matrix in order to identify the first two corners ($A$ and $B$). Then the third node with maximum theoretical distance to $A$ is identified and called $C$ where $SP(A,C) \simeq SP(B,C)$. The three nodes are placed on the border of a circle as their initial placement. The reason that only three corners are selected and not more or less is to avoid local-minima or layout folding in the drawing. For instance, if 4 corners are selected then there is a chance of a layout folding (see Figure 3.12b).

2. Selecting a randomly connected pair. The second option which is more general is just to find a randomly connected pair of nodes and start the drawing with them.

Another phase of the multilevel paradigm is the so called placement phase that is typically about where to introduce unvisited nodes on the screen. We have exploited Random Placer or RP which is a placement technique used by Bartel et al. [9]. At the beginning of each level, new nodes are identified. Each unvisited node is placed on the border of a small circle centered at its parent (a node which is directly connected to the unvisited node).

3.3.3 Optimization Solutions instead of the Multilevel Technique

As described earlier, the multilevel paradigm is a good solution in order to avoid local-minima and to decrease the running time. However, the multilevel technique is not always a good solution for small graphs or for some vulnerable types of graphs.
Figure 3.7 shows four different layouts for the same graph by a multilevel graph drawing algorithm [60].

Our solution for small graphs is drawing the graphs within a single level paradigm. That means all nodes are introduced into the drawing from the first iteration of the algorithm. Although most of the traditional single level algorithms start with a random initial node placement, our solution starts with a smart initial node placement using optimization algorithms.

As an alternative for the multilevel technique, we apply optimization techniques as a pre-processing step for small graphs. Most graph drawing algorithms, except those belonging to the MDS or spectral graph drawing approaches [81, 22], start with an initial random node placement which sometimes has a strong influence on the final
drawing. We found that the final layout can benefit from having a smart initial node placement instead of a random one [118, 117, 116]. We refer to our smart initial node placement by (SINP). After doing several different experiments on star-shaped graphs (it will be discussed in Section 5.2), we noticed that the chance of local-minima for such a graph topology is fairly high when non-multilevel algorithms such as FR [38] or KK [70] are applied. We also noticed that there is still a chance of local-minima (edge-crossings) when multilevel algorithms such as the Hu algorithm [60] are applied (see Figure 3.7). Therefore, it makes sense to apply optimization algorithms in order to have SINP instead of RINP (random initial node placement). Figure 3.7 shows a few drawings by \textit{sfdp} which is an implementation of Hu algorithm [60] in Graphviz [41].

In Chapter 5 we show how evolutionary algorithms as an optimization solution help with reducing the chance of local-minima. As shown by our work [117], star-shaped graphs are a vulnerable set of graph topologies which suffer from local-minima even if the graph is small. Our results [118, 117, 116] show that optimization solutions and specifically evolutionary algorithms can significantly help in this case. Although the optimization solutions are time consuming processes, it is still reasonable, in terms of running time, to apply them on small graphs.

3.3.4 Trigonometrical Approximation of Sine and Cosine

Most of the graph drawing algorithms require the calculation of line segments’ slope, where the line segments are created by pairs of nodes’ positions [38, 70, 122, 60, 82]. Calculating the \textit{sine} and \textit{cosine} requires obtaining the Euclidean distance between each pair of nodes. The Pythagoras equation is used in order to find the distance between two points.

We propose that the Euclidean distance between a pair of nodes’ positions ($p_{ni}$ and $p_{nj}$) can be replaced by Manhattan distance in order to reduce the computational burden and optimize the running time and still having an aesthetically acceptable layout. We call this new proposed technique (Trigonometrical Approximation of Sine and Cosine) \textbf{TASC} for short. The calculation is as follows:

1. Calculate the absolute value of the differences between the x and y coordinates ($px_{ni}$, $px_{nj}$) and ($py_{ni}$, $py_{nj}$) of two given points and assign them to $X$ and $Y$ respectively; where $px_{ni}$ and $py_{ni}$ are the x and y coordinates of $n_i$. This is equivalent to the calculation of Manhattan distance.
   \[X = |px_{ni} - px_{nj}|; \quad Y = |py_{ni} - py_{nj}|\]

2. Add $X$ to $Y$
Table 3.3 The time (seconds) for calculating sine and cosine with two different approaches.

<table>
<thead>
<tr>
<th>n</th>
<th>Using Euclidean distance</th>
<th>Using TASC 3.3.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00001</td>
<td>0.000008</td>
</tr>
<tr>
<td>100</td>
<td>0.00072</td>
<td>0.000493</td>
</tr>
<tr>
<td>500</td>
<td>0.00823</td>
<td>0.000623</td>
</tr>
<tr>
<td>1000</td>
<td>0.01184</td>
<td>0.002312</td>
</tr>
<tr>
<td>5000</td>
<td>0.06167</td>
<td>0.020133</td>
</tr>
<tr>
<td>10000</td>
<td>0.21501</td>
<td>0.064798</td>
</tr>
<tr>
<td>50000</td>
<td>5.14324</td>
<td>1.491735</td>
</tr>
<tr>
<td>100000</td>
<td>20.62015</td>
<td>5.986357</td>
</tr>
<tr>
<td>500000</td>
<td>519.78365</td>
<td>155.662585</td>
</tr>
</tbody>
</table>

As mentioned above, this method approximates the Euclidean distance between two points and consequently the sine and cosine. Therefore, there is an amount of error when approximation is used. The maximum possible error is when $|py_{ni} - py_{nj}| = |px_{ni} - px_{nj}|$ and the minimum error is when either $|py_{ni} - py_{nj}|$ or $|px_{ni} - px_{nj}|$ is very close to zero. Equation 3.2 shows the error and the calculation of sine and cosine using TASC where $\theta$ is the angle between $p_{ni}p_{nj}$ and x axis.

$$l_1 = \sqrt{(px_{ni} - px_{nj})^2 + (py_{ni} - py_{nj})^2}$$

$$l_2 = |px_{ni} - px_{nj}| + |py_{ni} - py_{nj}|$$

$$\text{sine}(\theta) = \frac{py_{nj} - py_{ni}}{l_2}$$

$$\text{cosine}(\theta) = \frac{px_{nj} - px_{ni}}{l_2}$$

$$\text{error} = 1 - \frac{l_1}{l_2}$$

The maximum error is 0.2928 which is $1 - \frac{\sqrt{2}}{2}$ and this is the case when $|py_{ni} - py_{nj}| = |px_{ni} - px_{nj}|$. We used the ordinary approach (Euclidean distance) and the approximation approach (see Section 3.3.4) in order to draw some graphs and compare some of them based on their layouts.

The running times for some sample data-sets are presented in Table 3.3. For a set of nodes $N$ there are $\frac{|N| \times (|N| - 1)}{2}$ pairs for which sine and cosine should be calculated. We generate $|N|$ points with different coordinates and applying the Euclidean distance and the TASC approximation algorithm.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

3.3.5 The Concentric Theoretical Distance Algorithm (CTD)

The CTD algorithm starts either with smart initial node placement (SINP) or follows the multilevel technique. If the graph size is small, around 100 nodes, the SINP results in a good output with reasonably acceptable running time. The CTD algorithm starts with all nodes when SINP is applied. The maximum number of iterations has an inverse relation to the size of the graph. A smaller number of iterations is needed for a larger graph (a graph with a large number of nodes). For instance, for a graph with around 3000 nodes, 50 iterations are needed, while for a graph with 50 nodes, 100 iterations are needed. This is because of the nature of the concentric approach in which nodes get adjusted quicker with a higher number of nodes. Since CTD does not minimize any stress or energy, the termination of the algorithm for small graphs is usually based on the number of iterations \( c \times |N| \) where \( c \) is a small number less than 5, and for large graphs (when the multilevel paradigm is applied) it takes as long as all nodes are introduced into the drawing plus \( \log |N| \times 5 \) iterations. We show the two approaches of CTD in regards to how to initiate the algorithm.

1. **SINP**: The output of the SINP method is the input of the CTD algorithm.

    The advantage of CTD is having a very simple implementation compared to other theoretical distance graph drawings algorithms such as Stress majorization or KK. CTD unlike most of the other force-directed graph drawing algorithms, does not need to find the physical distance between pairs of nodes in order to use them as stress or energy. CTD is an iterative schema and at each iteration the new positions for nodes are updated for the next iteration. The pseudo code of CTD using SINP is illustrated in algorithm 1.
Algorithm 1 Concentric Theoretical Distance (CTD) Algorithm using SINP

Read Graph G

\[ N = G.SIZE \] # The number of nodes
\[ D = D(G) \] # All Shortest Path Calculation
\[ SINP(G) \] # Finding a smart initial node placement

for \( (t = 0; t < c \times |N|; t++) \) do
  for \( (i = 0; i < |N|; i++) \) do
    \( x = 0; y = 0; \)
    \( z = 0; \) # In case of 3D
    for \( (j = 0; j < |N|; j++) \) do
      \[ x = x + \frac{px_{nj} - px_{ni}}{|p_{nj} - p_{ni}|} \times D[i][j] \]
      \[ y = y + \frac{py_{nj} - py_{ni}}{|p_{nj} - p_{ni}|} \times D[i][j] \]
      \[ z = z + \frac{pz_{nj} - pz_{ni}}{|p_{nj} - p_{ni}|} \times D[i][j] \]
    end for
    \( px_{ni} = x; \)
    \( py_{ni} = y; \)
    \( pz_{ni} = z; \)
  end for
end for

As mentioned earlier, using SINP is more appropriate for smaller graphs as SINP is an expensive process if the graph becomes larger. Small graphs, especially star-shaped graphs, benefit a lot from smart initial node placement (SINP) [116].

The theoretical distances for small graphs can be normalized in order to get a layout closer to its intrinsic topology or even to get rid of the distortion around the borders. Figure 3.8 shows a series of graphs with two different layouts for each graph. Layouts 3.8a, 3.8c and 3.8e are drawn using the actual theoretical distances while layouts 3.8b, 3.8d and 3.8f are drawn using normalized distances. Figure 3.9 shows another example that contains two layouts, one with normalized distances (see Figure 3.9a) and another with the actual distances (see Figure 3.9b). The natural logarithm of each of the distances are calculated in order to normalize them.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

(a) A layout of a graph with 39 nodes using distance matrix.

(b) A layout of a graph with 39 nodes using normalized distance matrix.

(c) A layout of a graph with 51 nodes using distance matrix.

(d) A layout of a graph with 51 nodes using normalized distance matrix.

(e) A layout of a graph with 26 nodes using distance matrix.

(f) A layout of a graph with 26 nodes using normalized distance matrix.

Fig. 3.8 Three different graphs with two drawings for each graph.
Fig. 3.9 Two different layouts for the same graph (40 × 40 mesh). The first figure is drawn using normalized distances (natural logarithm) and the second figure is drawn using the actual distances before normalizing.

2. MVN: Multilevel Vertex Neighboring based CTD.

CTD can adopt the multilevel technique in order to optimize the running time and to reduce the chance of getting trapped into a local-minimum. The traditional multilevel techniques that coarsen the nodes are not suitable for our algorithm. This is because CTD makes use of distance matrix and it is difficult to introduce theoretical distances between coarsened nodes. Therefore, our proposed multilevel technique does not coarsen the nodes. Using the MVN technique in CTD shows better results when dealing with larger graphs. MVN helps both the running time and the problem of local-minima, therefore it is a good solution for large graphs to be visualized by employing MVN in CTD. As mentioned earlier, there are two slightly different approaches for the MVN technique in terms of selecting the initial nodes. Either the corners of a graph or a random pair of nodes are selected. In both cases, the initially selected nodes are positioned around a circle. A hierarchical tree is created in which the initially selected nodes are at the top level of that hierarchy and are labeled as visited nodes. Then the unvisited neighbors of all visited nodes become the next level and so on until all nodes are added (visited). Figure 3.10 shows an example of applying MVN on a small graph with 25 nodes. MVN starts with the three corners of the graph and at each level, the neighbors of visited nodes are added. The unvisited nodes in
Figure 3.10 are red-colored. The corresponding hierarchy tree of the graph with 25 nodes and its corners at the top level of the tree is shown in Figure 3.10.

(a) Five levels of the MVN technique starting from three corners.

(b) Hierarchy Tree of the mesh graph with 25 nodes in Figure 3.10a with three nodes as the corners.

Fig. 3.10 Different levels of applying MVN on a graph with 25 nodes.

The placement of nodes starts from the top level of the hierarchy. The initial nodes (corners) need to be positioned at the very first iteration. First, a circle
with radius 1 is considered and $2\pi$ or 360 is divided by the sum of the theoretical distances of these three nodes ($SP(A,B) + SP(A,C) + SP(B,C)$) and then the nodes are placed on the border of that circle according to their theoretical distances.

$$\text{Chunk} = \frac{360}{SP(A,B) + SP(A,C) + SP(B,C)}$$

$$\text{Polar.Cordinate}[A] = 0^\circ$$

$$\text{Polar.Cordinate}[B] = (\text{Chunk} \times SP(A,B))^\circ$$

$$\text{Polar.Cordinate}[C] = (\text{Chunk} \times SP(B,C))^\circ + \text{Polar.Cordinate}[B]$$

Figure 3.11 shows an example for a mesh graph (200 × 50) where its three corners are selected and placed on the border of a circle.

The reason that only three corners are selected but not more is to avoid local-minima. A triangle is the simplest convex shape which includes all the points. If we start with three corners of the graph then the other nodes are added more efficiently during the process with a high probability of avoiding local-minima (layout folding) but if for instance, four corners are selected then the initial placement might lead to a local-minimum (layout folding) as the process of initial corner placement is not smart. In Figure 3.12 two examples for the same graph are shown. The first example (Figure 3.12a) shows the drawing with three initial corners $\{A, B, C\}$ with their placement on a circle. In this case the next corner (node $D$), will be added in an appropriate place when the layout is being formed. This is because node $D$ is a descendant of one of the corners (in the hierarchy
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

In contrast, the example in Figure 3.12b starts with four corners which are initially placed on the border of a circle that potentially might cause an edge crossing.

![Diagram showing MVN technique with three corners and four corners.]

(a) MVN technique with three corners of the graph.

(b) MVN technique with four corners of the graph.

Fig. 3.12 Two examples of selecting different number of corners; 3 and 4. Selecting more than 3 corners increases the chance of local-minima and the layout folds.

Our experiments show that even if we place the three corners randomly on the border of a circle the chance of local-minima is very low. This is because three connected nodes (as a triangle) never make an edge crossing as long as two nodes do not overlap each other. The reason that we are not using less than three corners, e.g. two corners, is again to avoid local-minima. If the algorithm starts with only one pair of corners then there is still a chance for folding. Figure 3.13a shows an example of a graph with only one pair of corners, while Figure 3.13b is drawn by choosing three corners.
Once the corners are found they are labeled as visited nodes, leveled at 0 and positioned (see Figure 3.10). The MVN technique then proceeds to other levels with their corresponding nodes and eventually their positions. The MVN technique first labels all the nodes as unvisited nodes except the corners which have been already found and placed on the border of a circle. Then the children of all corners are found and labeled as visited and leveled at 1. This process is repeated for the newly added (visited) nodes in order to find their unvisited children. A child here means a node adjacent to a visited node from the previous level. Algorithm 2 shows the process of the MVN technique (see Figure 3.10).

The CTD algorithm starts with nodes at level 0 and performs one iteration in order to refine the positions of the nodes. Then it moves on to the next level and adds new nodes belonging to the next level and performs another iteration and so on until all the nodes are visited. This means that the total number of iterations is almost equal to the number of levels which is almost equal to $\frac{LT}{2}$ where $LT$ is the Largest Theoretical distance (diameter) of the graph. This is because the algorithm starts with at least 2 farthest nodes. The algorithm reserves some Extra iterations (Ei) when all nodes are introduced to the layout for further improvement. Ei is decided according to the size of the graph and the sparseness of the graph. Usually $Ei = \log |N| \times c$ where $c$ is a constant, most of
The time equal to 5. Algorithm 3 shows the CTD algorithm when MVN technique is incorporated.

### 3.3.6 Experimental Evaluation

In this section our results for both small graphs and large graphs are presented and compared to the *Stress Majorization (SM)* [39] technique. We compare our results to SM for the following reasons:

1. CTD and SM are both iterative scheme algorithms.

2. CTD and SM both draw graphs directly using their distance matrix unlike other algorithms such as Maxent [40].

3. One important reason that we are comparing our algorithm to SM, which is an iterative schema algorithm, is that in the case of real-time drawing the iterative schema algorithms such as SM, KK, FR and Hu [39, 38, 70, 60] need a minimum amount of re-calculation in order to refine the positions of those newly added nodes. Several algorithms in the class of MDS or spectral matrix decomposition (which include algorithms like CMDS, SDE or SSDE [22, 21])
Algorithm 3 Concentric Theoretical Distance (CTD) Algorithm using MVN technique

Read Graph G

\( N = G.SIZE \)

\( D = D(G) \) \# All pairs Shortest Path Calculation. This part is not part of our contribution

\( DR = D(G)^{\beta} \) \# Distance Raising; \( \beta \) is a positive value in the range \( 2 \leq \beta \leq 5 \)

\( MVN(G) \) \# Node leveling

Set Time at Max \((level) + Ei \) \# \( Ei = \log |N| \times c \)

\( S = \{\text{Corners}\} \) \# Initiating the list of visited nodes (the corners)

Position \( S \) at the border of a circle

Set \( \epsilon \) at min \# Initiate a variable with a very small value

for \( (t = 1; t < \text{Time}; t++) \) do

for \( (i \in S) \) do

\( x = 0; \ y = 0; \ z = 0; \) \# In case of 3D

for \( (j \in S) \) do

if \( (i \neq j) \) then

\( x = x + \frac{px_{ni} - px_{nj}}{\left|y_{nj} - y_{ni}\right|} \times D[i][j] \)

\( y = y + \frac{py_{nj} - py_{ni}}{\left|y_{nj} - y_{ni}\right|} \times D[i][j] \)

\( z = z + \frac{pz_{nj} - pz_{ni}}{\left|y_{nj} - y_{ni}\right|} \times D[i][j] \)

end if

end for

\( px_{ni} = x; \)

\( py_{ni} = y; \)

\( pz_{ni} = z; \)

end for

for \( (i \in |level.t|) \) do

\( P_{ni} = P_{ni,\text{parent}} + \epsilon \) \# Next level’s nodes placement

end for

\( S = S + \{\text{level.t}\} \) \# Introducing next level’s nodes to the algorithm

end for
exist which sometimes are faster than our proposed algorithm but they are not the best options when it comes to the real-time drawing. The ideal real-time drawing algorithm - when some nodes are added after the layout is completed- should maintain the current layout or make a minimum change in the current layout while adding the new nodes.

Our experiments show that the SINP approach for small graphs results in better layouts compared to MVN. Although SINP is a slower technique than MVN, small graphs would not be negatively affected by the running time. Therefore, for small graphs the SINP approach is used while the MVN is used for larger graphs. A list of large and well-known graphs have been collected from a repository created by Hu [61] and the running time (seconds) for the CTD algorithm using MVN is compared to SM and illustrated in Table 3.4. The corresponding graphs are visualized by both CTD and SM and shown in Figures 3.14 to 3.19. More visual results are presented in Appendix A.1.1 in Figures A.1 to A.6. The CTD algorithm using MVN brings two particular advantages over the Stress Majorization algorithm which is the improved version of Kamada and Kawai approach. The Stress majorization and Kamada and Kawai are the two force-directed algorithms that employ theoretical distance matrices in order to draw graphs. As Kamada and Kawai stated, a graph layout is nice if the geometric distances and theoretical distances correspond to each other [70].

As illustrated in Figures A.2a and 3.19a CTD draws Sierpinski very well and much faster than SM with completely clear borders. Figure 3.14a shows a very clear layout for the graph called monien-ukieme while SM in Figure 3.14b generates a folding which can lead to misinterpretation of the topology of the graph. Figure 3.14c shows a layout for the graph called boeing-mscd01440, while SM gets trapped into a local-minimum (see Figure 3.14d). A folding happens in Figure A.3d by SM, while CTD produced a layout in Figure A.3c without that folding. Figures 3.15a and 3.15b are two layouts of the same graph called gset48 (2D torus, 60-by-50) by CTD and SM, respectively; this graph has a topology similar to a doughnut. The SM algorithm does not show the circular shape of the graph (especially the internal circle) as much as the CTD shows it. There is a similar case with gset49 which is drawn by CTD and SM in Figures 3.15c and 3.15c, where a folding occurs in the layout by SM, and not by CTD. Both gset49 and gset48 are symmetrical graphs [61]. Figures 3.17a and 3.17b are the layouts for the graph called nasa2146 by CTD and SM, respectively. SM makes the layout look folded or twisted. This is most probably because of the density of the graph and also the RINP. In these types of situations, MVN with an appropriate selection of corners can help to avoid local-minima. Figures 3.18a and 3.18b also illustrate two layouts by CTD
and SM for the graph called \textit{rw5151}, both layouts are clear enough. The same feature (symmetry) can be observed in Figure 3.18c for the graph called \textit{tokamak-utm3060}. Two different layouts of this graph (\textit{tokamak-utm3060}) are illustrated in Figures 3.18c and 3.18d. The layout by CTD shows a symmetrical layout of the graph while the layout by SM does not show the symmetry.

Fig. 3.14 Two different graphs from Hu’s collection [61] with their details in Table 3.4.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

Fig. 3.15 The Gset48 and the Gset49 graphs from Hu’s collection [61] with their details in Table 3.4.
Fig. 3.16 Two different graphs from Hu’s collection [61] with their details in Table 3.4.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

Fig. 3.17 Two different graphs from Hu’s collection [61] with their details in Table 3.4.
Fig. 3.18 Two different graphs from Hu’s collection [61] with their details in Table 3.4.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

Another part of our evaluation is to observe the effect of the Trigonometrical Approximation of Sine and Cosine (TASC) (see Section 3.3.4) and evaluate its quality and running time. We experimented with a few selected graphs and evaluated their running times. We then generated the results first with the CTD applying TASC and second with CTD without TASC. The results for the running time are presented in Table 3.5. According to the running times that are presented in Table 3.5, TASC can save almost 15% of the total time spent by CTD when TASC is not applied. The visual results when TASC is applied are illustrated in Figures 3.20, 3.21 and 3.22. More visual results are presented in Appendix A.1.2 in Figures A.7 and A.8.

Although the difference between the two layouts, one with TASC and another without TASC are obvious, the results by TASC are still clear and expressive. For instance, the layout of the graph called shgy-41 by CTD using TASC in Figure 3.22d clearly shows the details of the graph, however the layout bends a bit and it is not as symmetrical as the layout without TASC by CTD in Figure 3.22c. This is also the case for the graph jagmesh9 in Figures A.8a and A.8b and for the rw5151 graph in Figures 3.22a and 3.22b. The layout for the graph called nasa2146 by CTD using TASC results in a very similar layout to the layout by CTD without using TASC (see Figures 3.21a and 3.21b).
Fig. 3.20 Two different graphs from Hu’s collection [61] with their details in Table 3.5 using Euclidean distance and TASC, respectively.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

(a) \textit{nasa2146} by CTD without TASC.

(b) \textit{nasa2146} by CTD using TASC.

(c) \textit{oberwolfach-filter2D} by CTD without TASC.

(d) \textit{oberwolfach-filter2D} by CTD using TASC.

Fig. 3.21 Two different graphs from Hu’s collection [61] with their details in Table 3.5 using Euclidean distance and TASC, respectively.
Next, we present our evaluation on small graphs when SINP is employed. Eight selected small graphs are visualized by CTD using SINP and their layouts - starting from the first iteration (SINP) until the last iteration - are illustrated in Figures 3.23a, 3.24a, 3.25a, 3.26a and Figures A.9a, A.10a, A.11a, A.12a in Appendix A.1.3.

The same graphs are visualized by SM and their layouts are illustrated in Figures 3.23b, 3.24b, 3.25b, 3.26b and Figures A.9b, A.10b, A.11b, A.12b in Appendix A.1.3. Two distinct features of CTD using SINP are the small number of iterations and avoidance of local-minima which are quite common for star-shaped graphs, (see Section 5.2).
For instance the layout by CTD in Figure 3.23b has no edge crossing while the layout for the same graph by SM in Figure 3.23b has 2 edge crossings.

(a) A star-shaped graph with 6 branches by CTD using SINP, 39 nodes and 45 edges.

(b) A star-shaped graph by SM, 39 nodes and 45 edges.

Fig. 3.23 Star-shaped graph.

(a) A mesh graph with 64 nodes and 112 edges.

(b) A mesh graph by SM, 64 nodes and 112 edges.

Fig. 3.24 Grid graph.
As mentioned earlier, CTD can also avoid local-minima by emphasizing the pairs with smaller theoretical distances. To emphasize the smaller theoretical distances, one can raise distances to the power of a constant number, e.g. \( d(n_i, n_j)^c \) where \( c \) is a smaller integer value like 2 and 3. Then pairs with longer distances become exponentially longer than pairs with smaller distance. This results in bringing the pairs with smaller theoretical distances closer to each other.

Raising the theoretical distances is applied only on a few initial iterations of the CTD algorithm and then the actual distances are replaced for the rest of the iterations. We observed that it is beneficial to apply this approach to star-shaped graphs. Two
examples of raising the theoretical distances on star-shaped graphs are illustrated in Figure 3.27 even without SINP.

Distance raising and distance normalization are two different things and should not be confused. By distance raising, we enlarge the differences between shorter theoretical distances and longer theoretical distances. Distance normalization is the opposite. Distance normalization is a method to reduce the gap between the shortest theoretical distances and the longest theoretical distances. Therefore distance raising, as an option, can happen at a number of initial iterations, and distance normalization only towards the end. The SINP method can still be used in order to create the initial node placement smartly.

Fig. 3.27 Two graph layout examples by CTD and raising the theoretical distances.
CTD and Real-Time

Our algorithm is suitable for real-time drawing as a minimum number of iterations is needed when new nodes are added into the drawing. Our preliminary observation about using the current layout in order to add new nodes shows that for each newly added node only 0.013 seconds is needed to refine the positions of all nodes for a graph with 2000 nodes. The visual result after adding 50 nodes to the current layout for the 20 × 100 mesh graph in Table 3.4 is shown in Figure 3.28. Initially, the new nodes are placed close to their neighbors then after a few iterations, the new layout is ready.

Fig. 3.28 The layouts for a 20 × 100 mesh graph before and after adding 50 new nodes to the graph.
3.3 CTD: Concentric Approach in Graph Drawing using Theoretical Distance

Table 3.4 Stress Majorization (SM) vs. CTD.
Ei indicates the number of extra iterations when all nodes are introduced by MVN. The presented time is only for the actual CTD and SM algorithms. The time for reading the file, producing the visual layout and calculating the all pairs shortest paths are excluded from the reported time below.

| Figure | Graph                        | |N| | |E| | Run Time(s) | Run Time(s) | Ei          |
|--------|------------------------------|---|---|---|-----------------|---------------|-------------|
|        |                              |   |   |   | SM              | CTD           |             |
| 1      | 3.14                         | 1866 | 3538 | 18.24 | 0.689 | log(N) × 5 |
| 2      | 3.14                         | 1440 | 22415| 17.71 | 0.772 | log(N) × 5 |
| 3      | 3.15                         | 3000 | 6000 | 59.75 | 2.475 | log(N) × 5 |
| 4      | 3.16                         | 1005 | 3808 | 5.85  | 0.557 | log(N) × 5 |
| 5      | 3.16                         | 1440 | 4032 | 7.11  | 0.568 | log(N) × 5 |
| 6      | 3.17                         | 2146 | 35052| 25.56 | 0.954 | log(N) × 2 |
| 7      | 3.17                         | 1668 | 4541 | 17.03 | 0.938 | log(N) × 4 |
| 8      | 3.18                         | 5151 | 15248| 300.49| 6.967 | log(N) × 3 |
| 9      | 3.18                         | 3060 | 28880| 87.70 | 1.745 | log(N) × 5 |
| 10     | 3.19                         | 3282 | 6561 | 36.12 | 3.825 | log(N) × 1 |
| 11     | A.1                          | 3120 | 15355| 105.42| 3.082 | log(N) × 3 |
| 12     | A.2                          | 1095 | 2187 | 5.70  | 0.350 | log(N) × 5 |
| 13     | A.2                          | 3136 | 6112 | 45.71 | 3.002 | log(N) × 3 |
| 14     | A.2                          | 1036 | 1868 | 6.90  | 0.604 | log(N) × 5 |
| 15     | A.3                          | 2851 | 15093| 125.82| 1.640 | log(N) × 5 |
| 16     | A.3                          | 2283 | 23227| 51.23 | 0.901 | log(N) × 5 |
| 17     | A.3                          | 1242 | 4592 | 9.03  | 0.605 | log(N) × 5 |
| 18     | A.4                          | 1138 | 3156 | 5.97  | 0.329 | log(N) × 5 |
| 19     | A.4                          | 1349 | 3876 | 13.97 | 0.601 | log(N) × 3 |
| 20     | A.4                          | 1224 | 5822 | 8.42  | 0.581 | log(N) × 5 |
| 21     | A.5                          | 1856 | 4942 | 33.03 | 1.211 | log(N) × 5 |
| 22     | A.5                          | 3466 | 10215| 32.77 | 2.666 | log(N) × 1 |
| 23     | A.5                          | 1141 | 3162 | 3.47  | 0.413 | log(N) × 5 |
| 24     | A.6                          | 2000 | 3880 | 43.14 | 1.978 | log(N) × 5 |
| 25     | A.6                          | 3169 | 8836 | 84.18 | 2.676 | log(N) × 5 |
Table 3.5 CTD using Euclidean distances (PYG: Pythagoras equation) VS CTD using TASC (Manhattan distance).
The running time is only for the actual drawing algorithm; the time for calculating the all-pairs-shortest-path is excluded. Other information is illustrated in Table 3.4

| Graph              | |N| |E| | Run-Time(s) | Run-Time(s) |
|--------------------|---|---|---|----------------|--------------|
|                    |   |   |   | CTD-PYG         | CTD-TASC     |
| ag-monien-grid2    | 3136 | 6112 | 3.002 | 2.763           |
| boeing-msc01440    | 1440 | 22415 | 0.772 | 0.662           |
| HB-dwt-1005        | 1005 | 3808  | 0.557 | 0.413           |
| hb-jagmesh9        | 1349 | 3876  | 0.601 | 0.524           |
| sierpinski         | 3282 | 6561  | 3.825 | 3.187           |
| lshp3466           | 3466 | 10215 | 2.666 | 2.143           |
| rw5151             | 5151 | 15248 | 6.967 | 5.869           |
| nasa2146           | 2146 | 35052 | 0.954 | 0.786           |
| Oberwolfach-filter2D | 1668 | 4541  | 0.938 | 0.835           |
| shyy-41            | 3120 | 15355 | 3.082 | 2.525           |

3.3.7 Conclusion

We introduced a new algorithm to draw undirected and connected graphs using their theoretical distances. To the best of our knowledge, this proposed algorithm (CTD) is the first multilevel algorithm which draws graphs using the all-pairs-shortest-path matrix directly. Our algorithm is comparable with the current algorithms which make use of theoretical distances in the class of force-directed algorithms. Although there are some algorithms in the class of spectral matrix decomposition such as SSDE or CMDS, which can compete with CTD in terms of running time, they do not belong to the class of force-directed algorithms and they have to be re-initiated when it comes to real-time drawing, plus the quality of the drawings is not the same [21].

CTD can embed the multilevel technique. However, for small graphs SINP can be used in order to avoid local-minima. Layouts of graphs by CTD regardless of their size also benefit from raising the distances at a number of initial iterations. As an option, layouts of graphs can be made clearer by normalizing their theoretical distances for some certain graphs. For instance, if the natural logarithm of distances is applied instead of the actual distances then the visual results are slightly different (see Figure 3.8). A general chart of the CTD algorithm and its features is shown in Figure 3.29.
Fig. 3.29 The general schema of the CTD algorithm in terms of dealing with different graph sizes.
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

3.4.1 Introduction

In this section, we introduce a new method for visualizing data containing multiple individuals when the pairwise similarity/dissimilarity between individuals is available. This method is applicable to multidimensional data. Multidimensional data is a type of data with several individuals, where each individual has multiple different attributes/features or dimensions [66, 17]. For example, a dataset of several different text documents is considered as a multidimensional dataset if each text document contains multiple features. Once the features of a text document are known one can create a vector space model (VSM) for that text document [86, 104, 98, 131]. Other techniques such as Cosine similarity [106, 64] are applied on the vector space models in order to derive the similarity/dissimilarity between different text documents and eventually create a dissimilarity matrix of all individuals (text documents).

One of the main challenges in the field of multidimensional data is to represent them in an easily-comprehensible way for human perception. Visualization can be used as a comprehensible means for analyzing the multidimensional data. Visualization is a graphical representation of abstract data that helps human for a better data analysis [100, 125, 120].

The main task in multidimensional data visualization is reducing the dimensionality of the data from a high number of dimensions to a lower number of dimensions that can allow visualization (2D or 3D) to be understandable for human perception [25, 121, 76, 18, 91, 28, 72].

In this section a new concentric-based algorithm (CMD) is introduced in order to visualize multidimensional data if the data is presented in the form of a dissimilarity matrix for the individuals in the data. CMD is similar to CTD in a sense that both algorithms work with a matrix which represents the differences between all pairs of nodes. The difference between a pair of nodes can be the distance/dissimilarity between the two nodes in a pair. The difference between a pair of nodes in CTD is typically represented as the theoretical distance between the pair of nodes and it reveals how theoretically far these two nodes are from each other. The difference between a pair of nodes in CMD, however, is the dissimilarity between the pair of nodes. In CMD we are not dealing with graphs so there is no concept of theoretical distance and
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

edges. Therefore, the dissimilarity can be any positive real value that describes the dissimilarity of a pair of nodes.

CMD lays out data individuals on the screen in accordance with the dissimilarities between them. This is achieved by applying concentric circles for all existing pairs of data entities over a number of iterations. CMD applies the idea of the multilevel approach and to the best of our knowledge, CMD is the first iterative schema for multidimensional data visualization that embeds a multilevel paradigm in order to first, accelerate the process of visualization and second, reduce the chance of chaos in visualization. In this work, we show the results of applying CMD on a few examples of text corpora and also on some other artificially generated data. We compare CMD to t-SNE [85], which is the state-of-the-art in this field, and also to the classical multidimensional scaling (CMDS), which is one of the pioneers in this field. Our evaluation is based on the layout observation according to some prior knowledge about the data.

Our proposed algorithm takes the dissimilarity matrix of all data individuals (an individual being a text document, node or any other type of data which can be represented as a point) and builds a picture of the whole dataset where each individual is represented by a node. Our algorithm differentiates potential clusters (groups of nodes with high similarity (low dissimilarity) between them) by distancing them from each other. Similar to the other force-directed methods, CMD runs over a number of iterations. There are several other works that visualize similar types of data such as CMDS [26]. CMDS takes the dissimilarity matrix of all pairs of data individuals and approximates their positions, typically in 2D. Another work by Sammon [105] (Sammon’s projection) works similar to CMDS but with less emphasis on the pairs with higher similarities and finally, a recent visualization method by Maaten et al. [85] which is the state-of-the-art in the field of multidimensional data visualization.

### 3.4.2 Datasets

In this section, we present the dataset which is used for this visualization technique. Two different datasets are used for this visualization technique; one is the artificial test dataset and the other one is the real-world dataset.

**Artificial Test Datasets**

Our approach to creating artificial test data is to generate the dissimilarity matrix directly. To do so, first of all, we assume that there is a set of data individuals $N$
Table 3.6 The dissimilarity matrix of a dataset containing 1600 individuals within four different clusters each cluster containing 400 individuals. The cells on diagonal line shows the range of dissimilarities between the individuals which belong to the same cluster and the other cells show the range of dissimilarities between the individuals from different clusters. Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Cluster Size</th>
<th>400 M1</th>
<th>400 M2</th>
<th>400 M3</th>
<th>400 M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>400 M1</td>
<td>15-25</td>
<td>55-65</td>
<td>75-90</td>
<td>75-90</td>
</tr>
<tr>
<td>400 M2</td>
<td>55-65</td>
<td>15-25</td>
<td>75-90</td>
<td>75-90</td>
</tr>
<tr>
<td>400 M3</td>
<td>75-90</td>
<td>75-90</td>
<td>15-25</td>
<td>55-65</td>
</tr>
<tr>
<td>400 M4</td>
<td>75-90</td>
<td>75-90</td>
<td>55-65</td>
<td>15-25</td>
</tr>
</tbody>
</table>

and then we create a $|N|\times|N|$ dissimilarity matrix with some predefined clusters. To construct a dataset of this type we assume there are $m$ clusters with a certain degree of dissimilarity between the individuals in each cluster and a certain degree of dissimilarity between the individuals in one cluster to those in other clusters. Table 3.6 shows an example of how our artificial data is designed. In this example, we create a dissimilarity matrix for a dataset with 1600 individuals. We create 4 different clusters by assigning different dissimilarity percentages. For instance the dissimilarity between two individuals that belong to cluster $M1$ is randomly assigned a number between 15% – 25% and the dissimilarity between two individuals one belonging to $M1$ and the other belonging to $M2$ is randomly assigned a value between 55% – 65%, and so on. To do this we artificially create 4 different clusters in which the individuals which belong to the same cluster have a lower dissimilarity (higher similarity) compared to others. By creating such a dataset we are aware of the dependencies and the details (similarity/dissimilarity) between individuals, so our judgment about the visual result becomes easier. For instance, if we artificially create two clusters with low dissimilarities between their individuals, then we expect the visualization to position these two clusters close to each other. Looking at the layout by having some prior information about the dataset is a measurement that Maaten et al. [85] have proposed. We also follow this idea.

In this experiment, we have constructed six different artificial datasets with different levels of dissimilarities between the groups of individuals. The dissimilarities between
Algorithm 4 Creating test data

Declare \( m \); \# number of groups.
Declare \( n \); \# number of individuals.
Declare an empty \( m \times m \) Matrix \( group.similarity \).

for \( p = 1 \) to \( m \) do
  for \( q = 1 \) to \( m \) do
    \[ \text{group.similarity}[p][q] = \text{average}(\sum_{i \in \text{group}[p]}^{n} \sum_{j \in \text{group}[q]} \text{Dis}[i][j]) \]
  end for
end for

Table 3.7 Details of the dissimilarities for the first dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups ( M1 )</th>
<th>( M2 )</th>
<th>( M3 )</th>
<th>( M4 )</th>
<th>( M5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M1 )</td>
<td>47</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>( M2 )</td>
<td>100</td>
<td>46</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>( M3 )</td>
<td>100</td>
<td>100</td>
<td>46</td>
<td>100</td>
</tr>
<tr>
<td>( M4 )</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>46</td>
</tr>
<tr>
<td>( M5 )</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

groups of individuals in each of these six datasets (each consisting of 5 groups of individuals) are shown in Tables 3.7-3.12. The dissimilarity between individuals which belong to the same group in these five datasets are around 46\%, while in the sixth one it is higher (see Table 3.12). The dissimilarities between different groups in Tables 3.7 and 3.12 are similar while they have different degrees of dissimilarities between individuals belong to the same group in those two tables. Note: the dissimilarities are always between \([0\% - 100\%]\), 100\% dissimilarity is equivalent to 0\% similarity and vice versa.

Table 3.8 Details of the dissimilarities for the second dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups ( M1 )</th>
<th>( M2 )</th>
<th>( M3 )</th>
<th>( M4 )</th>
<th>( M5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M1 )</td>
<td>46</td>
<td>99.1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>( M2 )</td>
<td>99.1</td>
<td>45</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>( M3 )</td>
<td>100</td>
<td>100</td>
<td>47</td>
<td>100</td>
</tr>
<tr>
<td>( M4 )</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99.1</td>
</tr>
<tr>
<td>( M5 )</td>
<td>100</td>
<td>100</td>
<td>99.1</td>
<td>46</td>
</tr>
</tbody>
</table>
Table 3.9 Details of the dissimilarities for the third dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>46</td>
<td>84</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M2</td>
<td>84</td>
<td>46</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M3</td>
<td>100</td>
<td>100</td>
<td>47</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>47</td>
<td>85</td>
</tr>
<tr>
<td>M5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>85</td>
<td>47</td>
</tr>
</tbody>
</table>

Table 3.10 Details of the dissimilarities for the forth dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>46</td>
<td>70</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M2</td>
<td>70</td>
<td>46</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M3</td>
<td>100</td>
<td>100</td>
<td>45</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>46</td>
<td>70</td>
</tr>
<tr>
<td>M5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>70</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 3.11 Details of the dissimilarities for the fifth dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>47</td>
<td>58</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M2</td>
<td>58</td>
<td>46</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M3</td>
<td>100</td>
<td>100</td>
<td>46</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>46</td>
<td>57</td>
</tr>
<tr>
<td>M5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>57</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 3.12 Details of the dissimilarities for the sixth dataset with 2000 individuals and 5 different groups (400 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>7</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M2</td>
<td>100</td>
<td>8</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M3</td>
<td>100</td>
<td>100</td>
<td>8</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>M5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>8</td>
</tr>
</tbody>
</table>
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

Table 3.13 Details of the dissimilarities for the dataset used for 3D visualization with 1000 individuals and 5 different groups (200 individuals per group). Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>46</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M2</td>
<td>100</td>
<td>47</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M3</td>
<td>100</td>
<td>100</td>
<td>47</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>M4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>47</td>
<td>100</td>
</tr>
<tr>
<td>M5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>47</td>
</tr>
</tbody>
</table>

Our proposed algorithm is a multidimensional data visualization technique for representing data in both 2D and 3D spaces. To illustrate the usefulness of our algorithm for 3D visualization we create another dataset which consists of 1000 artificial individuals in five different groups with maximum dissimilarities between groups. The details of this dataset are shown in Table 3.13.

Real-World Dataset

We use four different sets of documents that have been collected from the website of the BBC. The first set consists of 2225 documents from BBC News published in the period 2004-2005 and covering stories in five different topical areas: (business, entertainment, politics, sport, and tech). The documents are distributed as follows. Business has 510 documents, entertainment has 386 documents, politics has 417 documents, sport has 511 documents and tech has 401 documents.

The second set consists of 737 documents from BBC Sport corresponding to sports news articles from 2004-2005 in five topical areas: (athletics, cricket, football, rugby, and tennis). Athletics has 101 documents, cricket has 124 documents, football has 265 documents, rugby has 147 documents, and tennis has 100 documents. These datasets are used for another type of experiment by Greene [49].

The third set consists of 990 documents from online chats. There are two different types of chats: normal chat and victim-predator chat. The victim-predator chats are broken down into two parts: victim and predator. Therefore 3 different clusters of documents are created: victim, predator and normal.

The forth set consists of 1500 documents all from online chats. There are two different types of chats: teenager-celebrity and victim-predator chat. The victim-predator chats are broken down into two parts: victim and predator and the same
for teenager-celebrity. Therefore there are 4 different clusters of documents: victim, predator, celebrities and teenager.

**Dataset Preparation**

To prepare the aforementioned datasets, one needs to create a dissimilarity matrix which shows the relations between each pair of documents. To do this, document term-frequency matrix is required to be generated. The concept of term-frequency originally comes from the study by Spärck Jones [69]. It was initially named term specificity and later on has become known as inverse document frequency, or IDF [107].

IDF is based on the number of documents in the collection of documents (corpus) being searched which contain (or are indexed by) the term. The idea is that a term which occurs in many documents should not be a good discriminator to rely on, and should be given less weight than one which occurs in a few documents.

The documents are represented using the vector-space model (VSM) in which each document \( (doc) \) is considered to be a vector in the term-space. In its simplest form, each document in a corpus \( N \) of \( |N| \) documents is represented as document term-frequency \( DTF = (TF_1 \times IDF_1, TF_2 \times IDF_2, \ldots, TF_{|N|} \times IDF_{|N|}) \) where \( TF_i \) is the frequency of the \( i_{th} \) term in the document and \( IDF_i \) is the inverse document frequency of the \( i_{th} \) term obtained by \( IDF_i = \log_{|DF_i|} |N| \) where \( DF_i \) (document frequency) is the number of documents that contain the \( i_{th} \) term [34].

This weighting schemes class is known in general as \( TF.IDF \), which involves multiplying the \( IDF \) measure (possibly one of a number of variants) by a \( TF \) measure. This has proved an extraordinarily efficient method to find document similarity [107, 34].

**Creating Dissimilarity (Distance) Matrix**

We have brought our real-world datasets (see Section 3.4.2) which are not ready yet for our visualization algorithm into the form that VSM requires. As mentioned previously, the dissimilarity between each pair of documents needs to be calculated for our proposed visualization algorithm. The commonly used approach to calculating the similarity between two documents \( doc_i \) and \( doc_j \) is cosine similarity expressed in Equation 3.3, where \( \cdot \) denotes the dot-product of two vectors. The distance matrix of the document term frequency matrix \( DTF \) \( (doc) \) is produced using Equation 3.3. Our proposed algorithm takes in dissimilarities between pairs of individuals, therefore the similarities have to be converted into dissimilarities. The cosine similarity only generates values in the range \([0, 1]\), therefore, all the similarities are subtracted by 1 in order to generate the dissimilarities between pairs of individuals (see Equation 3.4).
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

\[ Sim(doc_i, doc_j) = \cos(\theta) = \frac{\vec{doc}_i \cdot \vec{doc}_j}{|\vec{doc}_i| \times |\vec{doc}_j|} \]  

(3.3)

\[ Dis(doc_i, doc_j) = 1 - Sim(doc_i, doc_j) \]  

(3.4)

3.4.3 Concentric-Approach Based Visualization for Multidimensional Data using Dissimilarity matrix

Overview
Let \( n_1, n_2, \ldots, n_n \in N \) be \( |N| \) individuals (individuals can be a set of text documents). The input data for the proposed algorithm (CMD) is a \( |N| \times |N| \) dissimilarity matrix which represents pairwise dissimilarities of \( |N| \) individuals. CMD uses the idea of concentric circles in which the origin of the Cartesian space would be placed almost at the gravity point of the final layout. CMD is similar to CTD, however, unlike CTD there is no connectivity between individuals in the data. CMD employs the multilevel technique for the same reasons as CTD does. Accelerating the process plus reducing the chance of local-minima entrapment are the two reasons that the multilevel technique is employed in CMD as well. Since there is no edge or shortest path between any pair of individuals the multilevel technique in CMD is different from the one we proposed for CTD.

Algorithm
This algorithm is an iterative algorithm within the multilevel paradigm. To initiate the algorithm, the most centric node with the minimum sum of the dissimilarities to all other nodes is found and placed at an arbitrary location and it becomes a visited node. At the next level a few other unvisited nodes, again with the minimum sum of dissimilarities, are identified and placed near to the visited nodes closest to them. This process continues until all the nodes are visited. A similar technique to CTD is applied here in order to reduce the chance of local-minima by raising the dissimilarities to a positive power. An optional parameter \( I \) is used in the CMD algorithm in order to enlarge the difference between the smallest dissimilarity and the largest dissimilarity. CMD like CTD does not optimize any stress or energy of the layout. It uses the concentric approach to refine the positions of nodes (see Section 3.2). Therefore, the number of required iterations does not depend on the stress of the layout. The multilevel technique needs \( \frac{|N|}{L} \) levels, where \( L = \log |N| \), in order to introduce all the
nodes gradually into the drawing, that means \( \log |N| \) nodes are added at each level. CMD ends after \( R \) iterations where \( R = \frac{|N|}{\log |N|} + Ei \) where \( |N| \) is the number of nodes. \( Ei \) stands for Extra iterations when all the nodes are visited and introduced to the drawing. \( Ei \) is usually equal to \( \frac{|N|}{3} \). Our experiment shows that \( R = \frac{|N|}{\log |N|} + Ei \) is sufficient in order to get a layout that outlines the structure of the graph clearly when the multilevel technique is embedded.

The overall proposed algorithm is straightforward to implement. Algorithm 5 shows the pseudo-code of our proposed algorithm which contains the part for initial vertex placement as well.
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

**Algorithm 5** Interactive Multidimensional Data Visualization Algorithm

Read a $|N| \times |N|$ Dissimilarity matrix $D$ of a set of individuals $N$.

Declare an empty $Visited$ list $V$.

Declare an empty $SD$ list.

Find the sum of dissimilarities for all nodes and insert them into $SD$.

Find the most centric node $n_c$. # A node with the minimum sum of dissimilarities with all other nodes

Add $n_c$ into the $Visited$ list.

Declare the parameter $I$ # For raising the dissimilarities. $I$ is 1 by default

Accommodate $n_c$ at the center

Initiate the number of iterations $R$.

Initiate $Counter$ at 0.

**while** $Counter \leq R$ **do**

Declare $Ms = 0$

**while** $Ms \leq \log |N|$ **do**

**if** $n_i \notin Visited \&\& SD[i]$ is min **then**

place $n_i$ on a very small circle centered at $n_c$.

Add $n_i$ into $Visited$ list.

$n_c = n_i$

$Ms +=$

**end if**

**end while**

**for** all $i \in N$ **do**

Initiate $x$ at 0.

Initiate $y$ at 0.

Initiate $z$ at 0. # if 3D

**for** all $j \in N$ **do**

**if** $n_i \neq n_j \&\& n_i \in Visited \&\& n_j \in Visited$ **then**

$x = x + (Dis[n_i][n_j])^I \times \frac{px_{n_j} - px_{n_i}}{|px_{n_j} - px_{n_i}|}$

$y = y + (Dis[n_i][n_j])^I \times \frac{py_{n_j} - py_{n_i}}{|py_{n_j} - py_{n_i}|}$

$z = z + (Dis[n_i][n_j])^I \times \frac{pz_{n_j} - pz_{n_i}}{|pz_{n_j} - pz_{n_i}|}$ # if 3D

**end if**

**end for**

$px_{n_i} = x$;

$py_{n_i} = y$;

$pz_{n_i} = z$; # if 3D

**end for**

$Counter++$

**end while**
As shown in Algorithm 5, unlike the most of iterative algorithms, the positions of nodes at each iteration are not accumulated by positions from the previous iteration. In the next sections, the experimental results by CMD, t-SNE [85] and the classical MDS [67] are discussed in detail.

3.4.4 Visual Results

In this section, the visual results of CMD for several different datasets is presented, compared and contrasted with t-SNE and the classical MDS. t-SNE does not convert the dissimilarities into distances between nodes but it converts them into probabilities. Thus measuring the Euclidean distances and comparing them to dissimilarities (stress) is not a good method of comparison. Looking at the layout by having some information about the dataset in advance is a measurement that Maaten et al. [85] state in their work. In this work, we look at the layouts and discuss the layout based on the characteristics of the dataset. Running time can be another criterion in order to measure different visualization approaches.

Figure 3.30 shows three visualizations for a dataset generated by Algorithm 6. The data which is used in Figure 3.30 is a test dataset which contains 2000 nodes and 4 different groups A, B, C, D in which the dissimilarity between each pair of nodes is generated randomly within two different ranges. Algorithm 6 shows the corresponding pseudo code. As is shown in Algorithm 6, the range of dissimilarities between pairs of nodes from two different groups is twice larger than the range of dissimilarities between pairs of nodes belong to the same group. Data visualization should be able to reflect this feature in the visual representation by placing nodes in appropriate locations.

An obvious feature in the CMD visualization in Figure 3.30 is that four separate groups of data are clearly shown and separated, while the layout by t-SNE is not as clear. CMDS also shows very clear distinguished groups, however, the similarities between nodes within each group is too high compared to the similarities between individuals from different groups. Although the groups in t-SNE are separated enough without too much cluttering, if one color is used then t-SNE would not be able to accurately identify the borders between different groups (see Figure 3.31). In Figure 3.31, only CMD and CMDS are able to show 4 separate clusters.
Algorithm 6 Creating test data

Declare an empty set of data individuals $N$.
Divide $|N|$ by 4 in order to create 4 groups $A, B, C, D$ with exactly the same size.

for all $i \in N$ do 
    for all $j \in N$ do 
        if $(i \in A) \&\& (j \in A)$ then
            $Dis[i][j] = random.value \in [0, 1]$ 
        elseif $(i \in B) \&\& (j \in B)$ then 
            $Dis[i][j] = random.value \in [0, 1]$ 
        elseif $(i \in C) \&\& (j \in C)$ then 
            $Dis[i][j] = random.value \in [0, 1]$ 
        elseif $(i \in D) \&\& (j \in D)$ then 
            $Dis[i][j] = random.value \in [0, 1]$ 
        else 
            $Dis[i][j] = random.value \in [0, 2]$ 
        end if 
    end for 
end for
Concentric Approach in Graph Drawing using Distance Matrix

Fig. 3.30 Three different visualizations for the generated data by Algorithm 6.

Fig. 3.31 Three different visualizations for the generated data by Algorithm 6 with a unique color.

The six different data-sets which are represented in Tables 3.7-3.12 are visualized using three different approaches. The first dataset in Table 3.7 is comprised of 5 different groups with the maximum dissimilarity (no similarity) between different groups and with around 47% dissimilarity between individuals which belong to the same group. Figure 3.32 shows the visualization of the dataset in Table 3.7. Groups of 1 and 3 (blue and red) have no similarity but they are visualized close to each other by CMDS. CMD and t-SNE show a better and clearer layout in this case.
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

Fig. 3.32 Three different visualizations for the data in Table 3.7.

The second dataset in Table 3.8 has 5 different groups with around 46% dissimilarity between individuals which belong to the same group and small similarity (a bit less than the maximum dissimilarity) between groups 1 and 2 (red and green), similarly there is small similarity between groups 4 and 5 (pink and cyan), and the rest is exactly the same as in Table 3.7. Figure 3.33 shows three visualizations for the data in Table 3.8. CMDS in Figure 3.33 shows big similarity between groups 1 and 2 (red and green) while, in fact, there is only small similarity between them. t-SNE and CMD, however, show better and more reasonable distances between groups. The pink and cyan groups (4, 5) have a small similarity (less than 1%) and the pink and blue groups (3, 4) have no similarity. Therefore we can expect to see the average distance between the blue and pink groups (3, 4) is bigger than the average distance between the pink and cyan groups (4, 5); CMD can show this feature but t-SNE can not.

Fig. 3.33 Three different visualizations for the data in Table 3.8.
The third dataset in Table 3.9 has 5 different groups with around 46% dissimilarity between individuals in each group and a 84% dissimilarity between groups 1 and 2 (red and green), 85% dissimilarity between groups 4 and 5 (pink and cyan), and the rest is exactly the same as in Table 3.7. Figure 3.34 shows three visualizations for the data in Table 3.9. CMDS in Figure 3.34 shows the results in a way that cluster 1 overlaps cluster 2 and cluster 4 overlaps cluster 5. This implies that clusters 1 and 2 are very similar (low dissimilarity) while the dissimilarity is 84%.

We observed that t-SNE behaves strangely when there is one group which retains maximum dissimilarity to other groups while other groups have fewer dissimilarities between each other. Groups 1 and 2 (red and green) have a certain degree of dissimilarity to each other and so do groups of 4 and 5 (pink and cyan) while group 3 (blue) has no similarity to anyone. The t-SNE in Figure 3.34 shows a strange layout for group 3 (blue). By looking at the CMD layout in Figure 3.34 one can say that the distance between groups 1 and 2 (red and green) and the distance between groups 4 and 5 (pink and cyan) are smaller than the distance between group 3 (blue) and any other group.

Group 3 (blue) in the t-SNE layout is formed strangely and looks like a snake. It implies that the nodes at the head of the snake have large dissimilarities with the nodes at the tail of the snake. The average dissimilarities between individuals in group 3 are 47% with standard deviation at around 8%. As Maaten et al. [85] mentioned, t-SNE gives different layout every time as a result of random initial node placement. To guarantee that the strange results are not due to the fact of random initial node placement, we ran the t-SNE several times and we got the similar strange layout every time. CMD, however, makes use of the multilevel paradigm but not all the time the results are exactly the same. There is a stage in CMD which is called Placement that uses a random method to place nodes around a visited node. The difference between different layouts out of different runs is minor and it can be easily ignored.
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

(a) CMD. (b) t-SNE. (c) CMDS.

Fig. 3.34 Three different visualizations for the data in Table 3.9.

The fourth dataset in Table 3.10 has 5 different groups with around 46% dissimilarity between individuals in each group, 70% dissimilarity between groups 1 and 2 (red and green), 70% dissimilarity between groups 4 and 5 (pink and cyan), and the rest is exactly the same as in Table 3.7. Figure 3.35 shows three visualizations for the data in Table 3.10. CMDS in Figure 3.35 shows a maximum similarity between groups 1 and 2 and groups 4 and 5 that is similar to Figure 3.34. Group 3 (blue) has no similarity with any other group. t-SNE in Figure 3.35 shows again the same strange behavior (snake shape) as in Figure 3.34. Group 3 (blue) which has no similarity to any other group is visualized in a strange way, looks like a snake, between two pairs of groups (1 and 2) and (4 and 5). By observing the layout generated by CMD in Figure 3.35 we can say that the distances between groups 1 and 2 (red and green) and between groups 4 and 5 (pink and cyan) are much smaller than the distance between group 3 (blue) and any other group; this reflects the similarities in Table 3.10.

(a) CMD. (b) t-SNE. (c) CMDS.

Fig. 3.35 Three different visualizations for the data in Table 3.10.
The fifth dataset in Table 3.11 has 5 different groups with around 46% dissimilarity between individuals in each group, and 58% dissimilarity between groups 1 and 2 (red and green) and 57% dissimilarity between groups 4 and 5 (pink and cyan). The rest is exactly the same as in Table 3.7. Figure 3.36 shows three visualizations of the data in Table 3.11. The same strange situation for group 3 (blue) as in Figures 3.34 and 3.35 happens in Figure 3.36 too, however the larger similarities (lower dissimilarity) between groups 1 and 2 and groups 4 and 5 are clearly observed compared to Figures 3.34 and 3.35 for both t-SNE and CMD. CMDS again shows a maximum degree of similarity between groups 1 and 2 and groups 4 and 5.

Fig. 3.36 Three different visualizations for the data in Table 3.11.

The sixth dataset at Table 3.12 has 5 different groups with around 7% dissimilarity (very close to the minimum dissimilarity) between individuals in each group and a maximum degree of dissimilarity between any two groups. Figure 3.37 shows three visualizations for the data in Table 3.12. CMDS shows tight dependencies between the individuals in each group. The distances between different groups do not reflect the data in Table 3.12.

In contrast, t-SNE shows almost an even distribution for individuals in each group with almost equal distance between pairs of groups. However, the distances between individuals which belong to the same group do not reflect the large similarity (small dissimilarity) in Table 3.12. CMD shows tight dependencies between individuals which belong to the same group and almost equal distances between different groups that reflect the data in Table 3.12.
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

Fig. 3.37 Three different visualizations for the data in Table 3.12.

To show the usefulness of CMD for 3D visualization, we use the seventh dataset in Table 3.13 which is quite similar to the first dataset in Table 3.7 with 1000 individuals. A few snapshots from different angles are shown in Figure 3.38.

Fig. 3.38 Three different views of the visualization of the data in Table 3.13.

The next part of our visual evaluation is to evaluate our real world data. Four different real world datasets have been used in this work (see Section 3.4.2). The first dataset is BBC news with five different topics: Business (510 documents), entertainment (386 documents), politics (417 documents), sport (511 documents) and tech (401 documents). The dissimilarities between these individuals (documents) are obtained by considering each individual as a vector in the term space by calculating the document term-frequency (DTF) as explained in Section 3.4.2. Then, cosine similarity using Equations 3.3 and 3.4 are applied in order to find the dissimilarities in range \([0, 1]\) which is the same as in our artificial datasets. Figure 3.39 shows three different visualizations
for the BBC news dataset. Judging the usefulness of any visualization approach is challenging for a real dataset without additional information about the data. The different types of news are colored differently but it is still difficult to say how precise they are separated or how well they are positioned. To fairly evaluate the results we come up with a table similar to the ones we have employed for the artificial datasets. To do that the average dissimilarities between all the documents, categorized in different topics, are measured and summarized in Table 3.14. The last row in the table shows the sum of the dissimilarities for each group to all other groups. The smallest average dissimilarities are for group 5 (Tech) in cyan color. It can be observed that Tech news is spread among other news, especially in the layouts by CMD and CMDS (see Figure 3.39).

Table 3.14 Details of the dissimilarities for the BBC news dataset with 2225 documents and 5 different groups; 1: Business, 2: Entertainment, 3: Politics, 4: Sports, 5: Tech. Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.831</td>
<td>0.929</td>
<td>0.902</td>
<td>0.93</td>
<td>0.89</td>
</tr>
<tr>
<td>M2</td>
<td>0.929</td>
<td>0.931</td>
<td>0.786</td>
<td>0.922</td>
<td>0.905</td>
</tr>
<tr>
<td>M3</td>
<td>0.93</td>
<td>0.922</td>
<td>0.825</td>
<td>0.908</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>0.89</td>
<td>0.905</td>
<td>0.905</td>
<td>0.908</td>
<td>0.822</td>
</tr>
<tr>
<td>Sum</td>
<td>3.651%</td>
<td>3.686%</td>
<td>3.66%</td>
<td>3.681%</td>
<td>3.608%</td>
</tr>
</tbody>
</table>

(a) CMD.  (b) t-SNE.  (c) CMDS.

Fig. 3.39 Three different visualizations for the BBC news dataset.
3.4 CMD: Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix

The second real world dataset we used in our study is BBC sports news. There are five different groups. Athletics has 101 documents, cricket has 124 documents, football has 265 documents, rugby has 147 documents and tennis has 100 documents. The same thing (obtaining the average dissimilarities between all the documents categorized in different groups) is done for BBC sports news and the results are summarized in Table 3.15. In Table 3.15 the groups of Athletic and Tennis have low dissimilarity between them (red and cyan) (M1 and M5). In the layout by CMDS the cyan group is almost inside the red group and the same thing is observed in the layout by CMD.

Table 3.15 Details of the dissimilarities for the BBC sport news dataset with 737 documents and 5 different groups; 1: Athletics, 2: Cricket, 3: Football, 4: Rugby, 5: Tennis. Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.754</td>
<td>0.904</td>
<td>0.911</td>
<td>0.903</td>
<td>0.877</td>
</tr>
<tr>
<td>M2</td>
<td>0.904</td>
<td>0.728</td>
<td>0.908</td>
<td>0.870</td>
<td>0.890</td>
</tr>
<tr>
<td>M3</td>
<td>0.911</td>
<td>0.908</td>
<td>0.823</td>
<td>0.888</td>
<td>0.898</td>
</tr>
<tr>
<td>M4</td>
<td>0.903</td>
<td>0.870</td>
<td>0.888</td>
<td>0.764</td>
<td>0.887</td>
</tr>
<tr>
<td>M5</td>
<td>0.877</td>
<td>0.890</td>
<td>0.898</td>
<td>0.887</td>
<td>0.758</td>
</tr>
<tr>
<td>Sum</td>
<td>3.596</td>
<td>3.573</td>
<td>3.606</td>
<td>3.549</td>
<td>3.553</td>
</tr>
</tbody>
</table>

(a) CMD.  
(b) t-SNE.  
(c) CMDS.

Fig. 3.40 Three different visualizations for the BBC sports news dataset.

The third real world dataset that is used in our study is derived from some chat documents. There are three different groups of documents. Victims has 460 documents, predators has 460 documents and ordinary talks has 70 documents. The same thing
(obtaining the average dissimilarities between all the documents categorized in different groups) is done for the chat and the results are shown in Table 3.16. Table 3.16 indicates that the highest dissimilarities are between groups of ordinary talks and predators, (blue and green). This feature is observed in the CMD and CMDS layouts in Figure 3.41.

Table 3.16 Details of the dissimilarities for the chat data-set with 990 documents and 3 different groups; 1: Victims, 2: Predator, 3: Normal talks. Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>0.948</td>
<td>0.955</td>
<td>0.979</td>
</tr>
<tr>
<td>$M_2$</td>
<td>0.955</td>
<td>0.930</td>
<td>0.986</td>
</tr>
<tr>
<td>$M_3$</td>
<td>0.979</td>
<td>0.986</td>
<td>0.954</td>
</tr>
<tr>
<td>Sum</td>
<td>1.934</td>
<td>1.941</td>
<td>1.965</td>
</tr>
</tbody>
</table>

(a) CMD. (b) t-SNE. (c) CMDS.

Fig. 3.41 Three different visualizations for the Chat dataset.

The fourth real world dataset used in our study is a chat dataset as well. There are four different groups of documents. Victims has 460 documents, predators has 460 documents, celebrities chats has 70 documents and teenager has 510 documents. The same thing (obtaining the average dissimilarities between all the documents categorized in different groups) is done for the chat and the results are presented in Table 3.17. The predator group (red) has the lowest dissimilarity sum to other groups. This can be observed in the layout by CMD. M1 and M2 (red and green) have the maximum similarity and this feature is observed in both CMD and t-SNE layouts in Figure 3.42.
3.4 CMD: **Concentric Approach to Multidimensional Data Visualization using Dissimilarity Matrix**

Table 3.17 Details of the dissimilarities for the chat dataset with 1500 documents and 4 different groups: 1: Predators, 2: Victims, 3: Celebrities 4: Teenagers. Values in the table represent dissimilarity percentages.

<table>
<thead>
<tr>
<th>Groups</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.934</td>
<td>0.943</td>
<td>0.987</td>
<td>0.981</td>
</tr>
<tr>
<td>M2</td>
<td>0.942</td>
<td>0.892</td>
<td>0.990</td>
<td>0.986</td>
</tr>
<tr>
<td>M3</td>
<td>0.987</td>
<td>0.990</td>
<td>0.964</td>
<td>0.986</td>
</tr>
<tr>
<td>M4</td>
<td>0.981</td>
<td>0.986</td>
<td>0.986</td>
<td>0.968</td>
</tr>
<tr>
<td>Sum</td>
<td>2.91</td>
<td>2.919</td>
<td>2.963</td>
<td>2.953</td>
</tr>
</tbody>
</table>

(a) CMD. (b) t-SNE. (c) CMDS.

Fig. 3.42 Four different visualizations for the Chat dataset 3.17.

### 3.4.5 Conclusion

We propose a new approach in order to visualize high dimensional data into a lower dimensional environment using the concentric approach, (see Figure 3.3). The results from two other approaches (t-SNE and classical MDS) are contrasted with the proposed approach. Our experiment is divided into two different types of datasets (artificial and real-world). We devise the artificial dataset with some predefined features and we expect to see those features. CMD is successful in showing those features. It is not very easy to compare the layouts of real-world datasets. The results of our proposed algorithm show that our algorithm can produce results as strong as the t-SNE and for some cases even better than t-SNE. The running time of CMD is much better than t-SNE. To visualize the dataset presented in Table 3.7, CMD takes 18.067 seconds while t-SNE takes 5.54 min and CMDS takes 10.484 seconds, the same amount of
time is needed for visualizing the other datasets of the same size. All the running times are obtained on the same system. To measure the running time and generate the coordinates for t-SNE and CMDS the R package [97] is used. The running time is calculated only for the actual algorithms. The time for reading the input file, coloring and visualizing the coordinates are excluded from the running time.
Chapter 4

The Concentric Approach to Graph Drawing using Edge-Lists

4.1 Introduction

In this chapter we propose a model which has two algorithms (Sync and Burst) inspired by the concentric approach that is proposed in Chapter 3. The concentric approach utilizes multiple concentric circles with different radii corresponding to the theoretical distances between pairs of nodes.

The concentric approach can be applied when the input data is an adjacency matrix. An adjacency matrix for unweighted graphs gives a minimum amount of information about graphs’ components (nodes and edges). It indicates if a given pair of nodes are connected by an edge or not. In other words, the theoretical distance or shortest path for each pair of nodes \( SP(n_i, n_j) \), when the adjacency matrix is applied, is either \( SP(n_i, n_j) = 1 \) for adjacent nodes or \( SP(n_i, n_j) > 1 \) otherwise. This is represented as either 1 for directly connected nodes and 0 otherwise. We decided to name these two theoretical distances as near distance (for adjacent nodes) and far distance (for non adjacent nodes or for nodes such that their theoretical distance is greater than 1). The idea of using two circles for the two available distances (near and far), surrounds the layout in a circular area especially for graphs that are not very large. The reason that the layout seems surrounded by a circular area is the use of only one unique distance \( d \) (far distance) for all unconnected pairs of nodes. It does not matter how far two nodes theoretically are from each other; e.g. for two pairs of unconnected nodes with different theoretical distances \( (n_i, n_j) = 2 \) and \( (n_p, n_q) = 15 \) only one unique distance is considered (far distance). Therefore nodes with a higher theoretical distance from each other tend to stick around the boundary of a circle. There are a few works which
tend to produce circular shaped layouts. Bannister et al. [4] have introduced a third type of force called \textit{social gravity} which prevents nodes spreading too far out from the center of the layout. This force is stronger for nodes with a higher betweenness centrality. To do this, nodes are spread out evenly over a semi-circular-shaped area. This approach produces particularly efficient results (uniform node distribution on the drawing area) when applied to trees. Figure 4.1 shows two layouts by Bannister et al. [4], one with a classical force directed algorithm, Figure 4.1a and the other with employing social gravity, Figure 4.1b.

### 4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

In this section we introduce the CBD model, and the pseudo code of the algorithms along with some experimental results are presented. The BStress model is the closest to the algorithm proposed in this chapter however, we have approached the problem from a different angle and our results are slightly different from those of the BStress model. The similarity between CBD and BStress is that unlike the typical force-directed algorithms (single, or multilevel), neither BStress nor CBD make use of physical distances between nodes; e.g. both the algorithm by Fruchterman and Reingold [38] and the algorithm by Hu [60] use the physical distances between nodes in order to calculate forces.
The inspiration for the work here comes directly from our initial work on visualizing the synchronization on networks [119].

Generally speaking, synchronization is the behavior of more than one individual changing at a similar or the same rate. One of the well-known synchronization models is the one by Kuramoto [133]. This mathematical model considers individuals as a population of phase oscillators. Arenas et al. [2] have used this model on graphs in order to reveal the topological scales in networks by finding the phase differences. In that work, individuals are nodes and phases are their polar coordinates. Our initial work on graph drawing was to calculate the phase differences of all pairs of nodes and create a dissimilarity matrix that can be projected by CMDS techniques [119]. The results of our work have shown that visualizing the synchronization can create a general picture of a given graph; e.g. groups of highly interconnected nodes are revealed by reducing the edge-length. Although the synchronization model by Kuramoto is capable of revealing the topology of the graph without calculating any force or energy of the graph, it does not show enough details of the graph in the visualization. This is because synchronization brings highly interconnected nodes very close to each other so it might cause nodes’ overlapping.

To benefit from the idea of synchronization and also to avoid the problem of nodes’ overlapping we propose the CBD model. CBD visualizes any undirected graph by performing a number of iterations. CBD has two algorithms (sync and burst). The sync algorithm behaves similarly to the synchronization model by Kuramoto but much faster than that model in terms of running time. The burst algorithm is performed after the sync algorithm and it spreads nodes around into a circular shape.

CBD initially makes the nodes’ positions synced based on their connectivity; i.e. adjacent nodes are moved towards each other and tend to have a similar Cartesian positions (sync algorithm). The results of sync algorithm has a high node-density around the borders of the layout, (see Figure 4.2). Although the sync algorithm produces a layout with high density around the borders, it still reveals the general picture of the layout. The results of sync algorithm then can be used as the input of burst algorithm in order to expand nodes around within a circular shape with a more uniform distribution.

4.2.1 Sync Algorithm

The sync algorithm is the first algorithm of the CBD model. As mentioned earlier it behaves similarly to the synchronization model by Kuramoto. The sync algorithm tries to allocate similar positions to adjacent nodes. As stated above, one of the differences
The Concentric Approach to Graph Drawing using Edge-Lists

Fig. 4.2 The result of the sync algorithm on a graph with 2559 nodes and 4092 links.

between the proposed sync algorithm and the synchronization model by Kuramoto is that the sync algorithm has a better running time compared with Kuramoto model. The sync algorithm makes use of the concentric approach to draw a graph. To do this, two sets of circles are considered for each single node at each iteration. For a given node, $n_i \in N$, the first set contains one single circle with radius $U$. The value of $U$ starts with 1 and it is incremented after each iteration by 1. The second set of circles contains $d$ circles where $d = \text{deg}(n_i)$ or the number adjacent nodes to $n_i$. Each adjacent node to $n_i$ has its own circle with radius $l$ where $l$ is the distance of the adjacent node to the origin of the Cartesian system.

All circles in both sets are centered at the origin of the Cartesian system. Figure 4.3 illustrates the process of creating concentric circles in order to perform the sync algorithm at iteration $t$ for node number 1 or the orange node.

Figure 4.3c shows how nodes’ positions are updated; each node $n_j$ adjacent to the node number 1 uses a circle with radius $l$ where $l$ is the distance of $n_j$ to the origin of the Cartesian system. The green colored nodes (node number 2 and 3) in the Figure 4.3a are the adjacent nodes to the orange node (node number 1) and the blue ones (nodes number 4 and 5) are the non adjacent nodes to the orange node. Let Figure 4.3b be the given layout of the graph in Figure 4.3a. Now the concentric circles
associated with the orange node (node number 1) of the given layout is created as follows in Figure 4.3c: A circle centered at the origin of the Cartesian system with radius $U$ is considered (blue circle) and all the line segments between the orange node and any other node is projected into the blue circle with exactly the same slope but resized according to the diameter of the blue circle.

![Diagram of a graph with 5 nodes, a random placement for the graph, and two different types of circles for connected nodes and all the nodes.](image)

(a) A graph with 5 nodes.

(b) A random placement for the graph in order to use concentric circles to sync the graph.

(c) Two different types of circles for connected nodes (green circles) and for all the nodes (connected and unconnected nodes, blue circle).

Fig. 4.3 Visual illustration of sync phase for the orange node in a graph with 5 nodes.

In this example there are two adjacent and two non-adjacent nodes to the orange node (node number 1) so four line segments are considered between the orange node and any other node (blue and red dotted lines in Figure 4.3c). These lines are then projected into the blue circle with exactly the same slope and resized according to the diameter of blue circle.

For instance, the blue dotted lines between the orange node (number 1) and the blue nodes (nodes number 4 and 5) in Figure 4.3c are moved and resized within the blue
circle and shown as blue solid lines. The same thing happens for the red dotted lines. The red dotted lines between the orange node and the green nodes (nodes number 2 and 3) are moved and resized within the blue circle and shown as red solid lines. The small pink nodes (⊕) now represent the temporary positions of the orange node (number 1).

Since the degree of the orange node (node number 1) is two, then two concentric circles centered at the origin of the Cartesian system are considered; one for node number 2 and the other for node number 3. The radius of each circle is equivalent to the distance between that node and the origin of the Cartesian system.

To update the position of the orange node (node number 1) two variables of $AV_{all}$ and $AV_{c}$ are introduced. The average positions of pink nodes (⊕) on the blue solid line circle is calculated and assigned to $AV_{all}$ and the average of green nodes is calculated and assigned to $AV_{c}$, then the average of $AV_{all}$ and $AV_{c}$ is set as the new position of the orange node. The algorithm 7 depicts the pseudo code of the first algorithm of our model for one iteration.

**Quad Tree**

Algorithm 7 has a straightforward implementation, however for very large graphs the running time grows exponentially. This is because at each iteration there is the N-body problem for which the complexity would be $O(|N|^2)$, (see Section 2.1). The first for-loop in the algorithm finds the slopes of the line segments between all pairs of nodes. Since there is no force or theoretical distance associated with $sine$ and $cosine$, one can make use of quad tree algorithm proposed by Barnes et al. [7] in order to reduce the complexity from $O(|N|^2)$ to $O(|N|\log |N|)$. Quad tree first surrounds the layout within a square. It then divides the square by four. If any sub-divided area contains more than one node then that area is divided by four again. This process recursively continues until each square contains only one node or no node. Figure 4.4 shows an example of how quad tree can be adopted to such algorithms.

To find the sum of $sine$ and $cosine$ for a given node, the quad tree process starts the investigation from the first level squares.

For instance, in order to find the sum of $sine$ and $cosine$ for the red node in Figure 4.4, $l$ and $d$ are calculated where $l$ is the distance between the red node and the center of the red square (this is one of the first level square) and $d$ is the diameter of the red square. If $\frac{d}{l} < \theta$ then only the $sine$ and $cosine$ of the angle between the green line (the line between the red node and the center of the red square) and the x-axis is calculated and multiplied by the number of nodes inside of the red square (5 in this
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

Algorithm 7 Iteration $t$ for the sync algorithm, of Concentric Binary Distance, CBD model, for a Graph $G = (N, E)$. $U$ is the radius of the blue solid line circle in Figure 4.3 ($U$ is initially set at 1).

*Initiate three sets of $X$, $Y$, $Z$ with size $|N|$*

for $(i = 0; i < |N|; i + +)$ do

\[ X[i] = \left( \sum_{j=0,i \neq j}^{N} (p_{x_j} - p_{x_i})_{t-1} \right) \times U \]

\[ Y[i] = \left( \sum_{j=0,i \neq j}^{N} (p_{y_j} - p_{y_i})_{t-1} \right) \times U \]

\[ Z[i] = \left( \sum_{j=0,i \neq j}^{N} (p_{z_j} - p_{z_i})_{t-1} \right) \times U \]

end for

*Initiate three sets of $CX, CY, CZ$ with size $|N|$*

for each $e \in E$ do

\{ each edge is an unordered pair of nodes $v$ and $u$ \}

\{ $px_{n_i}$, $py_{n_i}$ and $pz_{n_i}$ are the $x$, $y$ and $z$ coordinates of $n_i$ \}

\[ CX[e.v] + = (px_{e.v})_{t-1} \quad \# \text{ x coordinate of nodes} \]

\[ CY[e.v] + = (py_{e.v})_{t-1} \quad \# \text{ y coordinate of nodes} \]

\[ CZ[e.v] + = (pz_{e.v})_{t-1} \quad \# \text{ z coordinate of nodes} \]

\[ CX[e.u] + = (px_{e.u})_{t-1} \quad \# \text{ x coordinate of nodes} \]

\[ CY[e.u] + = (py_{e.u})_{t-1} \quad \# \text{ y coordinate of nodes} \]

\[ CZ[e.u] + = (pz_{e.u})_{t-1} \quad \# \text{ z coordinate of nodes} \]

end for

for $(i = 0; i < |N|; i + +)$ do

\[ (px_{n_i})_t = \frac{X[i]}{|N|-1} + \frac{CX[i]}{deg(n_i)} \quad \# \text{ $deg(n_i)$ is the number of connected edges to node $i$.} \]

\[ (py_{n_i})_t = \frac{Y[i]}{|N|-1} + \frac{CY[i]}{deg(n_i)} \]

\[ (pz_{n_i})_t = \frac{Z[i]}{|N|-1} + \frac{CZ[i]}{deg(n_i)} \]

end for
If $\frac{d}{l} \geq \theta$ then the red square is broken down into four smaller and same size squares and the new $l$ and $d$ are calculated for each of the new square. The same process happens for the other three first level squares and so on.

The quad tree (QT) process can improve the running time for large graphs however the quality of the graph layout will not be exactly the same compared with when QT is not applied. Figure 4.5 shows two layouts by the sync algorithm, the left layout is the result of using QT and the right layout is the result of using $N \times N$ calculations. This graph is Sierpinski graph with 1095 nodes and 2187 edges. The layout on the left does not have equal sides’ length while the layout on the right does. The running time when QT is applied is almost 3.3 times faster then the normal procedure ($N \times N$) for this example.

We apply the quad tree technique mostly in the sync algorithm. One of the features of the sync algorithm is that of bringing the nodes physically closer to each other. We discovered that if the quad tree technique illustrated in Figure 4.4 is applied in the sync algorithm there may be a problem with stack overflow. The sync algorithm tends to bring groups of highly interconnected nodes closer to each other, this might sometimes cause nodes’ positions overlapping. Thus the quad tree has to recursively create a large number of squares in order to separate nodes from each other and accommodate them.
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

Fig. 4.5 Two layouts of Sierpinski graph. The left figure is produced using QT and the right figure without using QT.

into a separate square. This phenomenon might cause stack overflow in the actual implementation; for instance the nodes inside of the black circles in Figure 4.6.

Our solution to this problem is to modify the quad tree technique in order to get rid of this problem. Instead of recursively dividing the main drawing area by four at each iteration in order to accommodate each single node in a unique square, we divide the drawing area into $S$ equal-sized squares. The number of squares is calculated as follows: $S = k^2$ where $k = \frac{|N|}{\log{|N|}}$. For instance, a graph with 1005 nodes is mapped into a 6 by 6 grid; see Figure 4.7a. For any given node there are two phases to be done: first the lines between the given node and all other nodes belonging to the same square are identified and the sine and cosine of the angles between those lines and the x-axis are calculated. Second the lines between the given node and the center of each single square (except the square accommodating the given node) and the number of nodes inside of that square are identified and the sine and cosine of the angles between each line and the x-axis are calculated and multiplied by the number of nodes contained in that square. For instance in Figure 4.7b for the given node (red node), first the sine and cosine of the angles between all green arrows and the x-axis are calculated. Second the sine and cosine of the angle between each blue arrow and the x-axis is calculated and multiplied by the number of nodes contained in the square that the
The Concentric Approach to Graph Drawing using Edge-Lists

Fig. 4.6 Nodes inside of the circles have a high location density and sometimes they are overlapping.

By applying this technique, the number of calculations for a set of nodes $N$ is decreased from $O(|N|^2)$ to $O\left(\frac{|N|^2}{\log |N|}\right)$. To do this we can get rid of the problem of stack overflow and approximate the calculation in a faster way.

(a) A graph with 1005 nodes. All nodes are divided into a grid 6 by 6.

(b) An example of the modified version of QT technique. The calculation of sine and cosine between the red colored node with all other nodes is shown.

Fig. 4.7 General view of our modified version of quad tree technique.
Initial Node Placement (INP) in Sync Algorithm

The synchronization algorithm brings all the highly interconnected nodes’ positions closer to each other and reduces the chance of local-minima. However there is still a possibility that the graph may get trapped into a local-minimum. Figures 4.8a and 4.8b are the results of the CBD synchronization algorithm on the graph which is clearly shown in Figure 4.8c.

(a) A layout after synchronization with an edge crossing.

(b) Another layout after synchronization without an edge crossing.

(c) Another layout after burst algorithm.

Fig. 4.8 Three layouts for a graph with 39 nodes.

This situation is more likely to happen for a certain types of graphs such as star-shaped graphs as discussed in Section 5.2. Again the reason that a local-minimum happens is because of the initial node placement choice which is usually chosen randomly. This problem is comprehensively addressed in Chapter 5 where we show how smart initial node placement can replace random initial node placement. Our solution in order to create a smart INP is to make use of evolutionary algorithms and a simulated
annealing algorithm as a preprocessing step for the sync algorithm in order to avoid local-minima and possibly also to reduce the number of iterations. Figure 4.9 shows two synchronization runs for the same graph one using random INP (Figure 4.9a) and the other one using smart INP (Figure 4.9b). As is shown in the Figure 4.9, smart INP does not only help to reduce the chance of local-minima but it also helps to reduce the number of iterations. For small graphs running time would not be the main concern but the edge crossings would be.

(a) Synchronization algorithm with random initial placement. The graph is trapped into a local-minimum.

(b) Synchronization algorithm with smart initial placement, using simulated annealing.

Fig. 4.9 Synchronization algorithm of the depicted graph in Figure 4.8.
Synchronization can be a great help for some types of graphs in order to reveal as much detail of the graph as possible, however not for all types of graphs. Figure 4.10 shows a few layouts which synchronization alone is able to reveal enough details of the graph while the layouts by sync in Figure 4.11a (top row) are not able to reveal sufficient details of the graph structure.

(a) A layout after synchronization for Sierpiński graph with 123 nodes.

(b) A layout after synchronization for $40 \times 40$ mesh graph.

(c) A layout after synchronization for Sierpiński graph with 1095 nodes.

(d) A layout after synchronization for a circle graph with 60 nodes.

Fig. 4.10 A few graph examples where the sync algorithm can reveal the structure of the graph in their layouts.
The Concentric Approach to Graph Drawing using Edge-Lists

(a) A few examples of graphs where the sync algorithm is not able to reveal the real structure of the graph. The top row shows the layouts by the sync algorithm and the bottom row shows the layouts for the same graphs using the Fruchterman and Reingold algorithm.

Fig. 4.11 A few examples of applying the sync algorithm on different graphs.

Sync-Algorithm Complexity

The sync algorithm has two different for-loops, (See Algorithm 7). The second for-loop is the major part of the sync algorithm and plays an important role in the drawing. The first for-loop goes through all pairs of nodes and the second for-loop only goes through adjacent pairs. The second for-loop brings highly interconnected nodes closer to each other in order to synchronize the nodes’ positions as much as possible. If the second for-loop runs alone the highly interconnected nodes can get very close to each other and possibly collapse. Therefore the first for-loop is introduced in order to act as a fine tuner and prevents nodes from collapsing or becoming too close to each other. In other words, the second for-loop synchronizes the nodes’ positions and the first for-loop controls their distance to some extent. Although this algorithm should have both for-loops, it is not necessary to run the first for-loop for every single iteration.

A similar work to the sync algorithm in a different context is developed by us [115]. In that work a new graph drawing algorithm is developed in order to optimize the
time complexity by eliminating the forces between all unconnected pairs of nodes. Two different forces, known as attraction and repulsion forces, are introduced only between connected pairs of nodes. That idea from our work is exploited here in order to optimize the running time while introducing our fine tuner and removing the force calculation. For some large graphs (over 1k nodes), like those graphs that are shown in Figure 4.10, the first for-loop is performed at most every $F$ iterations where $F = \sqrt{|N|}$. However for large and sparse graphs $F$ is reduced to $F = \log |N|$. The first for-loop is more time consuming than the second for-loop this is because the first for-loop goes through all pairs of nodes in order to compute the sine and cosine values that are generated by the angle between the line segments and x-axis, while the second for-loop is much simpler.

The question now is how many iterations does sync algorithm require to terminate. The main feature of our proposed approach was to get rid of the energy and stress minimization (optimization) as they are expensive types of processes. Therefore our approach is not able to deal with these techniques in order to decide the moment of termination. Although the BStress model makes use of stress in its formula, it does not calculate the physical distances between nodes in order to optimize the stress. The solution for the BStress model is to stop the process when $|p(t+1) - p(t)| \leq 0.001$ where $|p(t)|$ is the sum of all nodes’ placement at iteration $t$. We have also used the same idea but instead of checking each two subsequent iterations we check the value of $|p(t+1) - p(t)|/|p(t)|$ where $t$ is an iteration that performs only the second for-loop and $t + 1$ is an iteration that performs both for-loops.

4.2.2 Burst Algorithm

The second algorithm of our proposed model is called Burst which normally spreads the nodes out around but within a circular boundary. After the first algorithm (sync) those groups of nodes that are highly interconnected get so close to each other that the lengths between connected nodes, especially for nodes with smaller betweenness centrality, become very short. In contrast nodes with higher betweenness centrality are connected with longer edges to each other. The idea of the burst algorithm is not very different from the sync algorithm, the only distinguishing difference between them is about how connected nodes interact with each other. The first algorithm (sync) has a number of different concentric circles for a given node (depending on the degree of the node) while in the burst algorithm there is only one circle for the adjacent nodes with radius ($C$). Similar to the sync algorithm, there is another circle for only the unconnected nodes with radius ($U$). Figure 4.12 shows how the burst algorithm uses
bi-concentric circles in order to update the nodes positions to spread them almost evenly within a circular shape.

The radius of the circle for connected nodes \( (C) \) is always less than the radius of the circle for unconnected nodes \( (U) \). Note that both circles increase after each iteration at the same rate; our experiments show that the best value is \( C = \frac{U}{2} \). Although the burst algorithm spreads the nodes around within a circular drawing area, this might not be a desirable goal for all types of graphs especially for large graphs; e.g. skirt and gearbox [61]. For very large graphs the sync algorithm as an standalone algorithm can present a layout which reveals enough details of the graph.

\[ \text{Algorithm 8: Pseudocode for the burst algorithm for one iteration.} \]

![Fig. 4.12 Visual illustration of burst phase for the orange node in a graph with 5 nodes](image)

In Figure 4.12c the average of the cyan nodes (number 4 and 5) and the pink nodes (number 2 and 3) is used in order to update the position of the orange node (node number 1). Algorithm 8 shows the pseudocode of the burst algorithm for one iteration. Two concentric circles are introduced to the algorithm with two radii \( U \) and \( C \) where
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

$U > C$. Figure 4.12 shows an example of the burst algorithm to update a given node’s position (the orange node or node number 1 in Figure 4.12). The lines connecting node number 1 (the orange node) with all other non-adjacent nodes to node number 1 are indicated with solid red lines in Figure 4.12c. These lines then are projected into a circle with radius $R$ (the blue circle with a thicker border) while keeping the original slopes, the projected lines are shown as two dotted green lines. A similar thing happens for lines connecting node number 1 to all its adjacent nodes, they are shown as solid blue lines. These two lines also are projected into a circle with radius $C$ (the green circle) while keeping the original slopes. They are shown as two thin and dotted brown lines, (see Figure 4.12c). Nodes number 2, 3, 4 and 5 are the new copies of node number 1 (the one to be updated). The average of these four nodes (2, 3, 4 and 5) is the updated position for node number 1.

QT in Burst Algorithm

Algorithm 8 can also make use of QT in order to accelerate the process. However there is an issue with directly using QT for the first loop. As shown in Algorithm 8, the first for-loop goes only through the unconnected pairs while QT goes through all pairs of nodes, no matter if a given pair is adjacent or not. Therefore the slopes of those line segments that are created by two adjacent nodes are also calculated which should have not been calculated. Note: The first for-loop is only for non adjacent pairs. Our solution to this issue is using the second for-loop of the algorithm. In the second for-loop of the algorithm the sine and cosine of the angle between the line segments between $n_i$ and its adjacent nodes and the x-axis are calculated. The only thing which is needed is to use the calculated sine and cosine in order to deduct them from those extra calculated terms in the first for-loop by QT.

Although QT optimizes the N-body problem in terms of running time, the quality of the layout will not be as aesthetic as the traditional approach to the N-body problem. Figure 4.35 illustrates three different layouts for the Finance graph. One of these (Figure 4.35a) is the result of the sync algorithm alone and the other two on the left are the results of the sync and burst algorithms. The difference between the two layouts visualized by sync and burst (Figures 4.35b and 4.35c) is about the way the N-body problem is dealt with in the burst algorithm. Figure 4.35c is visualized by sync and burst algorithms and QT is used in both algorithms while Figure 4.35b is visualized by sync and burst algorithms and QT is only used in the sync algorithm. And finally Figure 4.35a is only visualized by the sync algorithm using QT. The running time between QT and the traditional approach is noticeable (see Table 4.4) while the visual
Algorithm 8 Iteration $t$ for the burst algorithm, of Concentric Binary Distance (CBD) model, for a Graph $G = (N, E)$ with a set of nodes $N$ and a set of edges $E$.

$R$ is the radius of the bigger circle (the blue circle) for only unconnected pairs of nodes and $C$ is the radius of the smaller circle (the green circle) for only adjacent pairs of nodes; (see Figure 4.12)

**Initiate three sets of $RX$, $RY$, $RZ$ with size $|N|$**

for $(i = 0; i < |N|; i + + )$ do

\[ RX[i] = \sum_{j=0, i\neq j}^{\frac{|N|}{p_{n_i} - p_{n_j}}} (\frac{p_{n_i} - p_{n_j}}{|p_{n_i} - p_{n_j}|})_{t-1} \times R \]

\[ RY[i] = \sum_{j=0, i\neq j}^{\frac{N}{p_{n_i} - p_{n_j}}} (\frac{p_{n_i} - p_{n_j}}{|p_{n_i} - p_{n_j}|})_{t-1} \times R \]

\[ RZ[i] = \sum_{j=0, i\neq j}^{\frac{N}{p_{n_i} - p_{n_j}}} (\frac{p_{n_i} - p_{n_j}}{|p_{n_i} - p_{n_j}|})_{t-1} \times R \]

end for

**Initiate three sets of $CX$, $CY$, $CZ$ with size $|N|$**

for each $e \in E$ do

\{ each edge is an unordered pair of nodes $v$ and $u$ \}

\[ CX[e.v] = (\frac{p_{x.e.v} - p_{x.e.u}}{|p_{x.e.v} - p_{x.e.u}|})_{t-1} \times C \]

\[ CY[e.v] = (\frac{p_{y.e.v} - p_{y.e.u}}{|p_{y.e.v} - p_{y.e.u}|})_{t-1} \times C \]

\[ CZ[e.v] = (\frac{p_{z.e.v} - p_{z.e.u}}{|p_{z.e.v} - p_{z.e.u}|})_{t-1} \times C \]

end for

for $(i = 0; i < |N|; i + + )$ do

\[ (px_{n_i})_{t} = \frac{RX[i]+CX[i]}{\sum_{i=0}^{n-1}} \]

\[ (py_{n_i})_{t} = \frac{RY[i]+CY[i]}{\sum_{i=0}^{n-1}} \]

\[ (pz_{n_i})_{t} = \frac{RZ[i]+CZ[i]}{\sum_{i=0}^{n-1}} \]

end for
results are not significantly different. Therefore QT can be used to deal with the N-body problem.

4.2.3 Discussion

The CBD model is a combination of both sync and burst algorithms, however the sync algorithm alone is able to produce clear layouts for most graphs. The algorithm runs over a number of iterations ($T$). The sync algorithm runs always first and the result of the sync algorithm (nodes’ positions) becomes the initial start point of the burst algorithm. The CBD model is depicted in Algorithm 9.

**Algorithm 9 CBD Algorithm**

```
Initiate Graph $G = (N, E)$ # Random or Smart
Initiate $C$ # Radius of the circle for connected pairs
Initiate $R$ # Radius of the circle for unconnected pairs
$T_1 = Time.Sync$
$T_2 = Time.Burst$
for ($t = 1; t < T_1; t + +$) do
   Increment $R$
   Perform Sync Algorithm
end for
for ($t = 1; t < T_2; t + +$) do
   Increment $R$
   Increment $C$
   Perform Burst Algorithm
end for
```

The CBD model spends more iterations on the sync algorithm and only a few iterations on the burst algorithm. The reason for spending fewer iterations on the burst algorithm is that the main task of graph drawing is done by the sync algorithm, so that the burst algorithm only spreads them around. The nature of the CBD model, if the burst algorithm is applied, is to visualize the graph within a circular shape. In the case of 3D visualization, graphs tend to be visualized close to the surface of a sphere. This means that the layout of a given graph is influenced by that regular shape (circular or spherical if the burst is applied). This is why not all the graphs can properly illustrate their structures. For instance a graph that has a square shape such as a $|N| \times |N|$ mesh loses its big perspective view if burst algorithm applied, Figure 4.13b shows the layout for a $40 \times 40$ mesh in a circular shape. The sync algorithm alone creates a layout close to the structure of the graph that reveals the details of the graph; (see Figure 4.13a). This means that the sync algorithm alone is able, for certain types
of graphs, to generate a meaningful and visually expressive layouts. The results of the CBD model can compete with the BStress model in terms of running time and also the quality of the layout. We believe that our algorithm is very similar to the BStress model for two reasons, first of all neither our model nor BStress make use of the physical distances between pairs of nodes to minimize the stress or energy of the layout. Secondly, both models are able to produce a layout in a circular drawing area. However as Koren et al. [82] mentioned in their work, there are certain types of graphs that can not be drawn in a circular drawing area.

The general motivation behind this work is to reduce the complexity of the graph drawing algorithms. For instance the multilevel algorithm proposed by Hu [60] needs several different side-methods; e.g. coarsening, HYBRID coarsening, applying cutoff radius, oct-tree, and quad-tree. In our algorithm we can reduce the number of side-functions and still have a good quality layout. Some of our results can even compete with the algorithm by Hu [60] in both quality and running time. Figures 4.15, 4.16, 4.17 and 4.18 show a few examples of applying Hu’s model on a few selected graphs. A comparison of the running times between the Hu model and the CBD model is presented in Table 4.1 and the chart illustrating the running time comparison is shown in Figure 4.14. In terms of running time the CBD model is close to the model proposed by Hu; for sparse graphs like lung2 Hu outperforms our model, but for more dense
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

graphs like pkustk10 and finance256 our model outperforms the Hu’s model. The visual comparison between the Hu’s model and the CBD model indicates an improvement (a more clear and expressive layout) by CBD over Hu’s model for some graphs. For instance, the finance256 graph is comprised of eight identical cycles connected to each other. The layout by the CBD model in Figure 4.35 illustrates 3 layouts, two by burst algorithm and one by only sync algorithm and all of them show clearly that there are 8 identical cycles. Although the layout by sync has different size for the cycles compared with the one by burst algorithm, both of them (sync and burst) show very similar size for the cycles. The layout for the same graph (finance256) by the Hu’s model in Figure 4.16a shows the general structure of the graph at an acceptable level however the cycles do not look very similar. The oberwolffach graph is visualized by both the CBD and Hu models in Figures A.27 and 4.15a respectively. The graph is comprised of a mesh with six different big holes around the border of the mesh. This structure (having six different holes around the border) is clearly indicated in the layout by CBD model (see Figure A.27) while the layout for the same graph by Hu does not illustrate the six different holes around the corner of the mesh. We believe that our results for the gset-g34 and plusk10 graphs by CBD in Figures 4.30 and 4.36 outperform the layouts for the same graphs by Hu in Figures 4.15b and 4.16b in terms of representing clearer layouts. For example the gset-g34 graph is a doughnut with a complete circle, the CBD model can show this feature (a complete circle), (see Figure 4.30) however the layout by Hu shows an oval instead of a complete circle inside of the doughnut, (see Figure 4.15b).

The layout of ag-monien-ubk-dual graph by CBD in Figure 4.29 shows an improvement over the layout by Hu in Figure 4.17b especially around the border of the layout. Our other comparison is the visual results of the Skirt graph by CBD and Hu models. Figures 4.39 and 4.18a show the layout of Skirt graph by CBD and Hu respectively. There are a few connections inside of each circle in the skirt graph which are not visible in the layout by Hu while they are visible by the CBD model.

Table 4.1 Details of the comparison running time between the model by Hu and CBD model for a few selected graphs.

<table>
<thead>
<tr>
<th>Model</th>
<th>Oberwolf</th>
<th>ag – monien ubk – dual</th>
<th>skirt</th>
<th>gset34</th>
<th>finance</th>
<th>pkustk10</th>
<th>lung2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>1.10</td>
<td>0.63</td>
<td>0.61</td>
<td>13.68</td>
<td>21.01</td>
<td>73.89</td>
<td>146.85</td>
</tr>
<tr>
<td>Burst</td>
<td>1.21</td>
<td>0.87</td>
<td>1.19</td>
<td>none</td>
<td>29.60</td>
<td>86.66</td>
<td>168.53</td>
</tr>
<tr>
<td>Hu</td>
<td>0.86</td>
<td>0.91</td>
<td>0.98</td>
<td>4.91</td>
<td>43.82</td>
<td>102.42</td>
<td>132.42</td>
</tr>
</tbody>
</table>
Fig. 4.14 The comparison results of running times between Hu algorithm and CBD model.

(a) Oberwolfach-filter2D with 1668 nodes and 4541 edges. Running time 0.86 sec using Graphviz.

(b) finance256 with 2000 nodes and 4000 edges. Running time 0.98 sec using Graphviz

Fig. 4.15 A few examples of applying Hu algorithm on different graphs.

Another motivation for developing such a model (sync and burst together) is to utilize the drawing area when several graphs are needed to be presented in a similar drawing area. Sometimes a graph with a long branch needs to occupy a big rectangular drawing area where most of the area is unused, therefore the need for making the drawing area into a more regular shape is sometimes necessary (see Figure 4.19).
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

(a) finance256 with 37376 nodes and 130560 edges. Running time 43.82 sec using Graphviz

(b) pkustk10 with 80676 nodes and 2114154 edges. Running time 102.42 sec using Graphviz.

Fig. 4.16 A few examples of applying Hu algorithm on different graphs.

(a) lung2 with 109460 nodes and 273646 edges. Running time 132.42 sec using Graphviz.

(b) ag-monien-ubk-dual with 1866 nodes and 3538 edges. Running time 0.91 sec using Graphviz.

Fig. 4.17 A few examples of applying Hu algorithm on different graphs.
Fig. 4.18 An example of applying Hu algorithm on a graph with 8882 nodes.

Fig. 4.19 Two layouts for the same graph using two different methods (CBD model and FR algorithm).
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

A circular drawing area can be converted into a square drawing area by keeping the original scaling using some geometric functions. Figure 4.20 shows how two trees are first visualized by CBD within a circular drawing area and then converted into a squared drawing area.

As mentioned previously, our model is comparable with the BStress model for two reasons, first of all because neither of them make use of physical distances between nodes (unlike traditional force directed algorithms) and secondly both algorithms are able to produce a layout within a 2D circular drawing area. BStress and CBD produce similar layouts for most graphs however for some specific graphs CBD is able to produce more clear layouts. For instance CBD, for some symmetrical graphs such as Wagner graph or Heawood graph [50], can generate a more expressive layout than BStress, (see Figures A.15 and 4.25).

Another advantage of CBD over BStress is in 3D drawing. CBD is able to visualize graphs over almost the surface of a sphere while 3D drawing by BStress is only able to visualize the graph in the normal 3D fashion, and not on the surface of a sphere. This kind of visualization might have potential application when a geographical data needs to be visualized on a sphere. CBD with a lower number of iterations on burst algorithm can also produce normal 3D visualization.

As mentioned previously, the idea of the first algorithm (sync) comes from our previous work [119]. In that work we have made use of a dynamical process (in that case synchronization dynamics) to visualize the behavior of the graph. The specific characteristic of the synchronization dynamics by Kuramoto [2] have shown that the highly interconnected nodes (a group of nodes which have more connectivity among themselves compared with the rest of the nodes) tend to get closer polar coordinates. The polar coordinates of nodes are then used to calculate the similarity matrix by making use of cosine similarity. By using the similarity matrix one can apply
multidimensional scaling approach in order to visualize them [119]. The synchronization approach (Kuramoto model [133]) works best for graphs with highly interconnected groups of nodes (groups of nodes with high density of connectivity). This is because a group of nodes with a high degree of interconnectivity among each other, tends to be synchronized faster [2]. Therefore sparse graphs do not benefit from the Kuramoto model. The sync algorithm works well with sparse graphs like trees as well. Figure 4.21 shows a few examples of applying the sync algorithm on trees. While the details and structure of the graphs in Figure 4.21 are not very clear, the main branches of the trees are clearly shown without any cluttering or crossing between them. The results of sync can be used as the initial start point of the burst algorithm or even any other force-directed algorithm.

3 Dimensional Visualization

Almost all of the force-directed graph drawing algorithms are adaptable for 3D drawing including BStress. CBD as a graph drawing algorithm is categorized into the class of force-directed and iterative graph drawing algorithms and is also adaptable for 3D drawing. One of the characteristics of our proposed algorithm is that CBD can visualize any graph on a surface of a sphere without making use of hyperbolic functions. The spherical surface drawing happens due to the use of the burst algorithm of our model.
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

The burst algorithm pushes all the nodes towards the border of a circle (sphere) by enlarging the radii of the far and near circles ($R$ and $C$) in the Algorithm 9.

**Implementation, Time Complexity and Running Time**

CBD as an iterative scheme has a better time complexity per iteration in comparison with the BStress model, however the required number of iterations in CBD is higher than BStress. The Table 4.2 and the Chart 4.22 show the details of the running time of a few selected graphs by CBD and BStress.

Table 4.2 Details of the running time for the BStress model and the CBD model on a few selected graphs.

<table>
<thead>
<tr>
<th>Model</th>
<th>nopoly</th>
<th>skirt</th>
<th>tuma2</th>
<th>finance256</th>
<th>pkustk10</th>
<th>ford2</th>
<th>gearbox</th>
<th>lung2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sync</strong></td>
<td>21.041</td>
<td>13.868</td>
<td>3.096</td>
<td>21.01</td>
<td>73.89</td>
<td>96.729</td>
<td>289.6</td>
<td>146.851</td>
</tr>
<tr>
<td><strong>burst</strong></td>
<td>22.875</td>
<td>0</td>
<td>3.353</td>
<td>29.60</td>
<td>86.66</td>
<td>0</td>
<td>0</td>
<td>168.537</td>
</tr>
<tr>
<td><strong>BStress</strong></td>
<td>26.733</td>
<td>38.586</td>
<td>3.289</td>
<td>188.2</td>
<td>1159.74</td>
<td>92.796</td>
<td>1838.2</td>
<td>513.613</td>
</tr>
</tbody>
</table>

The BStress model contains two expensive steps at each single iteration. First of all BStress computes the accumulated of all cosines and sines associated with the
angles associated with each node, \((\text{Acum cosine})\) for \(\text{cosine}\) and \((\text{Acum sine})\) for \(\text{sine}\); e.g. \(\text{Acum cosine}(n_i) = \left(\sum_{j=0}^{N} \left|\frac{p_x n_j - p_x n_i}{|p_x n_j - p_x n_i|}\right| \right)\) and \(\text{Acum sine}(n_i) = \left(\sum_{j=0}^{N} \left|\frac{p_y n_j - p_y n_i}{|p_y n_j - p_y n_i|}\right| \right)\). This has the complexity of \(\mathcal{O}(|N|^2)\), however by making use of Barnes-Hut scheme \([7]\) the complexity can be reduced to \(\mathcal{O}(|N| \times \log |N|)\). Secondly a \(|N| \times |N|\) matrix \((S)\) which is the summation of the Laplacian matrix \(L\) and another matrix, \(M\), which is a symmetrical matrix \((|N| \times |N|)\) and all the entries of the matrix are \(-1\) except the diagonal line of the matrix which is \(|N| - 1\). The summation of those two matrices \((M + \alpha L)\) is a symmetrical matrix \(S\); \(\alpha\) is a parameter which controls the lengths of edges. When \(\alpha\) has a large value then BStress behaves little like the sync algorithm in our model. The positions of the nodes are updated as given in Equation 4.1 in BStress:

\[
(M + \alpha L) \times x(t + 1) = \text{Acum cosine}(t)
\]

\[
(M + \alpha L) \times y(t + 1) = \text{Acum sine}(t)
\]

The complexity of the matrix multiplications in Equation 4.1 seems to be \(\mathcal{O}(|N|^2)\) but since the matrix \((M + \alpha L)\) is symmetrical and uniform then the multiplication can be performed with \(\mathcal{O}(|N| + |E|)\). The number of required iterations in BStress is very low and it mostly depends on the sparsity of the graph. In contrast our proposed algorithm has different complexity; CBD also makes use of the Barnes-Hut scheme in order to moderate the \(\mathcal{O}(|N|^2)\) into \(\mathcal{O}\left(\frac{|N|^2}{\log |N|} + |E|\right)\). The second for-loop with time complexity \(\mathcal{O}(|E|)\), in the sync algorithm (see Algorithm 7) is very fast to compute as there is no calculation like \(\text{sine}\) and \(\text{cosine}\) involved, the only calculation in the for-loop is the one that adds up the positions of neighbors. As has been noted before, the first for-loop in the sync algorithm has less effect on synchronization than the second for-loop therefore for most of the graphs the first for-loop can only be performed for a few iterations. However this is not the case in the burst algorithm as in the burst algorithm the two for-loops have to be performed for all iterations. Therefore the complexity of the sync algorithm is \(\mathcal{O}\left(\frac{|N|^2}{\log |N|} + |E|\right)\) per iteration and the complexity of the burst algorithm is \(\mathcal{O}\left(\frac{|N|^2}{\log |N|} + |E|\right)\) per iteration. The running time of the second for-loop in the sync algorithm that iterates \(|E|\) times is much lesser than the running time of the second for-loop in the burst algorithm that iterates \(|E|\) times as well. This is because the second for-loop in the sync algorithm has a very light computation while in the burst algorithm the computation is heavier. For most of the large graphs the number of iterations in the burst algorithm is \(b\) where \(b\) is a very low and almost irrelevant to the size of the graph. The total number of iterations is about \(c|N|\) where \(c\) is constant value less than 5.
4.2.4 Visual Results and Statistics

In this section our visual results along with the running times are presented. We divide our results into two different parts: 1) Small and Average Sized Graphs and 2) Large Graphs. This is because the running time for small and average graphs is very small and it is negligible so it is not an important measure to evaluate the work. Table 4.3 shows the results of some small-sized graphs with their layouts in Figures A.14, A.15, A.16, A.17 and A.18 in Appendix A.1 and 4.23, 4.24, 4.25 here in this chapter. The visual results of a few well known graphs with some certain types of topology such as the layouts of Heawood, Wagner, Peterson and Queen graphs show that CBD tends to clearly show the topology of the graph. This is not what other force-directed graph drawing algorithms such as FR or KK do. Most of the force-directed algorithms draw graphs in a way to decrease the edge lengths however for some graphs shorter edge length is not desirable. For instance in order to show the real structure of the Queen graphs\footnote{A Queen graph is a graph with $p \times q$ nodes in which nodes are simulated in $p \times q$ chessboard, each node behaves as a queen in chess game, if a queen beats another queen then there is a link between them otherwise not [Chvátal]} some edges remain longer than other in order to reveal the symmetry of the graph. The same thing happens in the Wagner graph where the four diameters of the semi-circle are longer than other edges but CBD is able to keep them long; this happens due to the first algorithm (sync) of our model.
Table 4.3 Details of some small graphs using CBD. *No.Sync* and *No.Burst* are the number of iterations spent on sync algorithm and burst algorithm respectively; (see Algorithm 7 and Algorithm 8)

<table>
<thead>
<tr>
<th>Graphs</th>
<th>N</th>
<th>E</th>
<th>No.Sync</th>
<th>No.Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star – Shaped</td>
<td>39</td>
<td>45</td>
<td>1 ×</td>
<td>N</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>123</td>
<td>243</td>
<td>2 ×</td>
<td>N</td>
</tr>
<tr>
<td>Tree</td>
<td>90</td>
<td>89</td>
<td>2 ×</td>
<td>N</td>
</tr>
<tr>
<td>Star – Shaped</td>
<td>33</td>
<td>69</td>
<td>2 ×</td>
<td>N</td>
</tr>
<tr>
<td>Heawood</td>
<td>14</td>
<td>21</td>
<td>5 ×</td>
<td>N</td>
</tr>
<tr>
<td>Wagner</td>
<td>8</td>
<td>12</td>
<td>6 ×</td>
<td>N</td>
</tr>
<tr>
<td>Peterson</td>
<td>10</td>
<td>15</td>
<td>5 ×</td>
<td>N</td>
</tr>
<tr>
<td>Queen8X8</td>
<td>64</td>
<td>728</td>
<td>3 ×</td>
<td>N</td>
</tr>
<tr>
<td>Queen9X12</td>
<td>96</td>
<td>1368</td>
<td>3 ×</td>
<td>N</td>
</tr>
</tbody>
</table>

(a) A layout by sync algorithm. (b) A layout by sync and burst algorithms. (c) A layout by BStress.

Fig. 4.23 Queen graph (8 × 12) with 96 nodes and 1368 edges. (See Table 4.3 number 9)
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

(a) A layout by sync algorithm.  
(b) A layout by sync and burst algorithms with 10 iterations.  
(c) A layout by BStress.

Fig. 4.24 Sierpinski graph with 123 nodes and 243 edges. (See Table 4.3 number 2)

(a) A layout by sync algorithm.  
(b) A layout by sync and burst algorithms.  
(c) A layout by BStress.

Fig. 4.25 Wagner graph with 8 nodes and 12 edges. (See Table 4.3 number 6)
Table 4.4 Details of some large graphs using CBD. The running time of the sync algorithm and sync & burst algorithms (*Total*) are reported separately.

| Figure | Graphs          | |N| | |E| | No. Sync | No. Burst | Total (sec) | Sync (sec) |
|--------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 1      | 4.26 | jagmesh8 | 1141 | 3162 | 2 × |N| | 7 | 0.435 | 0.32 |
| 2      | 4.27 | rw5151  | 5151 | 15248 | 1 × |N| | 3-7 | 3.102-3.596 | 2.23 |
| 3      | 4.28 | tree 2000 | 2000 | 1999 | 1 × |N| | 5-15 | 2.085-2.117 | 1.69 |
| 4      | 4.29 | ag – monienu kerbe1dual | 1866 | 3538 | 2 × |N| | 5-15 | 0.661-0.873 | 0.63 |
| 5      | 4.30 | gsetg34 | 2000 | 4000 | 2 × |N| | 10 | 1.192 | 0.61 |
| 6      | 4.31 | jagmesh1 | 936 | 2664 | 2 × |N| | 5-15 | 0.466-0.547 | 0.26 |
| 7      | 4.32 | jagmesh4 | 1440 | 4032 | 2 × |N| | 20 | 0.975 | 0.31 |
| 8      | 4.33 | nopoly | 10774 | 30034 | 3 × |N| | 2-5 | 22.875-24.012 | 21.04 |
| 9      | 4.34 | tuma2 | 12992 | 20925 | 0.3 × |N| | 5 | 3.353 | 3.09 |
| 10     | 4.35 | finance256 | 37376 | 130560 | 0.5 × |N| | 5 | 72.996 | 21.01 |
| 11     | 4.35 | finance256 | 37376 | 130560 | 0.5 × |N| | 5 | 29.607 | 21.01 |
| 12     | 4.36 | plastk10 | 80676 | 2114154 | 0.07 × |N| | 5 | 86.666 | 73.89 |
| 13     | 4.37 | hbdwt1005 | 1005 | 3808 | 3 × |N| | 5 | 0.493 | 0.47 |
| 14     | 4.38 | ford2 | 100197 | 322442 | 0.1 × |N| | 0 | None | 96.72 |
| 15     | 4.39 | skirt | 12597 | 91961 | 0.5 × |N| | 0 | None | 13.86 |
| 16     | A.19 | Mesh 20X100 | 2000 | 3880 | 1 × |N| | 5-15 | 0.445-0.684 | 0.35 |
| 17     | A.20 | agmonien | 2559 | 4092 | 3 × |N| | 5-15 | 2.050-2.381 | 1.94 |
| 18     | A.21 | ag – monien grid2 | 3136 | 6112 | 1 × |N| | 5-15 | 1.096-1.864 | 0.87 |
| 19     | A.22 | ag – monien stufe | 1036 | 1868 | 2 × |N| | 5-10 | 0.314-0.404 | 0.21 |
| 20     | A.23 | utm3060 | 3060 | 28880 | 2 × |N| | 5-10 | 3.105-3.252 | 2.94 |
| 21     | A.24 | wangswang2 | 3169 | 8836 | 2 × |N| | 5-15 | 2.160-3.68 | 1.97 |
| 22     | A.25 | Sierpinski gearbox | 1095 | 2187 | 5 × |N| | 5-15 | 0.965-1.139 | 0.94 |
| 23     | A.26 | Oberwolfach filter2D | 153746 | 4463329 | 8000 | None | 289.65 |
| 24     | A.27 | lung2 | 109460 | 273646 | 0.3 × |N| | 10 | 168.537 | 146.85 |

\(^2\text{Drawing finance256 by CBD without using QT}\)
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

(a) A layout by sync algorithm.  
(b) A layout by sync and burst algorithms.

Fig. 4.26 jagmesh8, nodes: 1141, edges: 3162. (See Table 4.4 number 1).

(a) A layout by sync algorithm.  
(b) A layout by sync and burst algorithms (3 iterations for burst).  
(c) A layout by sync and burst algorithms (7 iterations for burst).

Fig. 4.27 rw5151, nodes: 5151, edges: 15248. (See Table 4.4 number 2).
(a) A layout by sync algorithm.
(b) A layout by sync and burst algorithms (15 iterations for burst).
(c) A layout by sync and burst algorithms (5 iterations for burst).

Fig. 4.28 A tree, nodes: 2000, edges: 1999. (See Table 4.4 number 4).

(a) A layout by sync algorithm.
(b) A layout by sync and burst algorithms (5 iterations for burst).
(c) A layout by sync and burst algorithms (15 iterations for burst).

Fig. 4.29 ag-monien-ukerbe1, nodes: 1866, edges: 3538. (See Table 4.4 number 8).
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

(a) A layout by sync algorithm.

(b) A layout by sync and burst algorithms (10 iterations for burst).

Fig. 4.30 gset-g34, nodes: 2000, edges: 4000. (See Table 4.4 number 9).

(a) A layout by sync algorithm.

(b) A layout by sync and burst algorithms (5 iterations for burst).

(c) A layout by sync and burst algorithms (15 iterations for burst).

Fig. 4.31 jagmesh1, nodes: 936, edges: 2664. (See Table 4.4 number 10).
The Concentric Approach to Graph Drawing using Edge-Lists

(a) A layout by sync algorithm.  
(b) A layout by sync and burst algorithms (20 iterations for burst).

Fig. 4.32 \textit{jagmesh4}, nodes: 1440, edges: 4032. (See Table 4.4 number 14).

(a) A layout by sync algorithm.  
(b) A layout by sync and burst algorithms (2 iterations for burst).  
(c) A layout by sync and burst algorithms (5 iterations for burst).

Fig. 4.33 \textit{nopoly}, nodes: 10774, edges: 30034. (See Table 4.4 number 15).
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

(a) A layout by sync algorithm.

(b) A layout by sync and burst algorithms (5 iterations for burst).

Fig. 4.34 tuma2, nodes: 12992, edges: 20925. (See Table 4.4 number 16).

(a) A layout by sync algorithm.

(b) A layout by sync and burst algorithms without quad-tree technique (5 iterations for burst).

(c) A layout by sync and burst algorithms with quad-tree technique (5 iterations for burst).

Fig. 4.35 finance256, nodes: 37376, edges: 130560. (See Table 4.4 number 17 and 18).
Fig. 4.36 *plustk10*, nodes: 80676, edges: 2114154. (See Table 4.4 number 19).

(a) A layout by sync algorithm.  

(b) A layout by sync and burst algorithms (5 iterations for burst).

Fig. 4.37 *hb-dwt-1005*, nodes: 1005, edges: 3808. (See Table 4.4 number 24).

(a) A layout by sync algorithm.  

(b) A layout by sync and burst algorithms (5 iterations for burst).
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

Fig. 4.38 A layout for ford2 graph by sync algorithm, nodes: 100196, edges: 222246. (See Table 4.4 number 20).

Fig. 4.39 A layout for skirt graph by sync algorithm, nodes: 12598, edges: 91961. (See Table 4.4 number 22).

Table 4.4 includes the information for several large graphs visualized by CBD. For each graph the results of the sync algorithm and the burst algorithm are represented separately so that one can see the effect of the sync algorithm and the burst algorithm.
The Concentric Approach to Graph Drawing using Edge-Lists

on the graphs separately. The sync algorithm is the core of the CBD model. The sync algorithm - as it has been mentioned earlier - synchronizes the positions of connected nodes with each other which results in reducing the chance of getting trapped into a local-minimum. In the context of visualization, the sync algorithm can sometimes alone produce a clear and not necessarily a circular layout. For most graphs such as (rw515, Sierpinski, tuma2, wang-swang2, ag-montien-grid2, ag-montien-ukerbe1, finance256, gset-g34, jagmesh8) the sync algorithm alone is enough in order to represent a clear graph layout, however this is not the case for some other types of graphs such as trees and meshes. Some graphs with very special and complicated structures such as skirt or gearbox do not benefit from the burst algorithm therefore they are visualized only by the sync algorithm.

The running time of the sync algorithm is much smaller than the burst algorithm and this is because of two reasons:

1) Each node only averages its neighbors’ positions therefore there is no need to calculate any line segment length.

2) Each node calculates the accumulated sine and cosine between the angles associated with itself and all other nodes. This process can be performed by employing the adopted quad tree technique (see Section 4.2.1) which has been used by Barnes et al. [7] for a similar purpose, and moreover this step does not need to be performed at each single iteration.

The burst algorithm has a longer running time per iteration compared with the sync algorithm but the number of required iterations for burst is a lot fewer than the required number of iteration in the sync algorithm.

Figure 4.40 shows the layout of the finance512 graph. This graph has a similar topology to finance256 but with a higher number of connected circles. As shown in Figure 4.40 the BStress model is not able to show the topology and many details of the graph in an expressive way, or in other words, the layout does not easily imply the topology of the graph. The layout for finance256 by the CBD model in Figure 4.35 has a completely different structure and the topology of the graph is easily understandable. One major difference between CBD and BStress relates to the interior of the circle. BStress tends to fill in the interior of the circle even if by sacrificing the structure of the layout; e.g. layouts for the heawood graph and finance512 in Figures 4.40 and A.15. CBD in contrast visualizes the graphs in a circular shape, while outlining the structure of the graph to a greater degree than BStress, (see Figure 4.35 and A.15 for the finance256 and heawood graphs).
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

As has been mentioned earlier, CBD is able to visualize graphs on the surface of a complete sphere. As Koren have mentioned in their work [82], their algorithm like most of the other force-directed algorithms, is extensible into 3 dimensional visualization too. The advantage of our proposed algorithm over BStress is that CBD is able to visualize graphs also on the surface of a sphere in which those applications that need to map geographical data on the surface of a complete sphere would benefit of this type of drawing. To show that a graph is positioned around the surface of a complete circle we moved the 3D visualized graph and centered it at the origin of a 3 dimensional Cartesian system. Then the distance ($rc$) is found where $rc$ is the length of the closest node to the origin of the 3 dimensional Cartesian system. Then a complete sphere centered at the origin of the 3D Cartesian system with radius $rc$ is drawn. By rotating the visualized graph around the 3D scene one can observe how nodes are close to the surface of the sphere; i.e. we can observe if there are many nodes outside of the sphere and how far they are from the surface of the sphere. Figures A.30, A.31 and A.32 from Appendix A.1 and Figures 4.44 and 4.45 in this chapter show some examples of our results. We show three different views of each 3D layout once with CBD using a sphere to evaluate how far nodes are from a surface of a sphere and once with CBD without that sphere. (Note: there is no node inside of the sphere because the radius of the sphere is equal to the length of the nearest node to the origin of the 3D drawing area. Nodes are either on the surface or outside of the surface of the sphere; we use this method to show how well nodes are distributed around the surface of a sphere.) We
applied the same strategy (smallest sphere) for BStress in order to show the difference between the 3D layouts by CBD and BStress. The 3D visualization spends a bigger number of iterations on the burst algorithm compared to the sync algorithm. Normally the larger number of iterations for the burst algorithm makes the 3D visualization more spherical. Figures 4.41 and 4.43 show two visualizations for two graphs using different portions of the burst algorithm.

(a) Three different views of a 3D visualization for a graph with 39 nodes by the CBD model. (Fewer number of iterations is applied for the burst algorithm compared with the layout in Figure 4.41b).

(b) Three different views of a 3D visualization for a graph with 39 nodes by the CBD model. (A higher number of iterations is applied in the burst algorithm compared with the layout in Figure 4.41a).

Fig. 4.41 Star-Shaped graph with 39 nodes and 45 edges. Three different views of 3D visualization by the CBD model.
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

Fig. 4.42 Three different views of a 3D visualization for a tree with 90 nodes by the CBD model. (Fewer number of iterations is applied for the burst algorithm compared with the layout in the below Figure).

Fig. 4.43 Three different views of a 3D visualization for a tree with 90 nodes by the CBD model. (A higher number of iterations is applied for the burst algorithm compared with the layout in the above Figure).
(a) Two different views of a 3D visualization for a queen graph with 64 nodes by the CBD model using a sphere to show how close nodes are to the surface of the sphere.

(b) Two different views of a 3D visualization for a queen graph with 64 nodes by the CBD model without using sphere.

(c) Two different views of a 3D visualization for a queen graph with 64 nodes by the BStress model using a sphere to show how close nodes are to the surface of the sphere.

Fig. 4.44 A queen graph (8 × 8) with 64 nodes and 728 edges. (See Table 4.3 number 8)
4.2 CBD: Concentric Approach in Graph Drawing using Binary Distance

(a) Two different views of a 3D visualization for a graph with 33 nodes by the CBD model using a sphere to show how close nodes are to the surface of the sphere.

(b) Two different views of a 3D visualization for a graph with 33 nodes by the CBD model without using sphere.

(c) Two different views of a 3D visualization for a graph with 33 nodes by the BStress model using a sphere to show how close nodes are to the surface of the sphere.

Fig. 4.45 Two different views of a 3D layout for a tree with 33 and 69 edges by CBD and BStress.
4.3 Conclusion

In this chapter we presented a new model in order to visualize undirected graphs in 2D or 3D spaces using the concentric approach. Our presented model, CBD, has several characteristics which differentiate it from most other force-directed algorithms. First of all it does not use physical distances between nodes in order to draw graphs such as FR, KK and the model by Hu [60] or most of the other multilevel graph drawing algorithms. Another feature of CBD is that it uses two different algorithms: Sync and Burst, each of which produce a layout that can show the details of the graph to some extent. However for certain graphs such as ag-monien-grid2 graph in Figure A.21, sync alone can be enough, (Note: the burst algorithm has to come after the sync algorithm, that means the burst algorithm can not be applied alone). CBD is able to use a similar technique to the quad tree technique in order to accelerate the process of calculating all the necessary sine and cosine values associated with |N| nodes which normally has \(O(|N|^2)\) complexity.

The quad tree technique by Barnes et al. [7] has a minor issue with the sync algorithm. It might cause stack overflow due to the nodes’ position overlapping and this is why we developed a new technique inspired by the quad tree technique. The complexity of the modified quad tree technique is \(O\left(\frac{|N|^2}{\log |N|}\right)\). Our proposed algorithm is comparable with the BStress algorithm by Koren et al. [82] as both algorithms do not make use of physical distances between nodes and both tend to produce circular-shaped layouts. Although the number of iterations in BStress is lower compared with CBD, the running time of our algorithm can compete with BStress. This is because the running time of each single iteration is much smaller than the running time of each single iteration in BStress. Our proposed model can also compete with the model by Hu [60] in both running time and quality. We believe that our model constitutes a contribution to the field of graph drawing as it is a relatively fast method and produces readable layout according in keeping with the structure of the graph.
Chapter 5

Investigation into the Reduction of Local-minima using Optimization Algorithms

5.1 Introduction

The class of force-directed graph drawing algorithms is considered as probably the most successful approach to finding aesthetically pleasing layouts of general graphs in 2D [30, 78]. This is due to a number of factors which include the relative simplicity of implementation and reasonable running time. This is achieved by both emphasizing the structure of the graph (by placing highly interconnected nodes close to each other and avoiding unnecessarily long edges) and by the reduction in the number of edge crossings [95]. As empirically shown by Purchase [94], the number of edge crossings in a layout is one of the key indicators of its aesthetic qualities.

As discussed and analyzed in the previous chapters, the classical force-directed algorithms, such as the spring embedder of Eades [30], the algorithm of Fruchterman and Reingold (FR) [38], and the algorithm of Kamada and Kawai [70] require an initial layout which is typically produced by assigning random coordinates to nodes, for reasons of efficiency. They then consider the input graph as a mechanical system with forces of attraction and repulsion between nodes, and in a series of iterations lower the total energy of the system by changing the placement of the nodes. One of the weaknesses of classical force-directed algorithms is that the total energy of the system can be trapped at a local-minimum and the final layout might contain avoidable edge crossings. The choice of an initial layout as a starting point of the algorithm can
Investigation into the Reduction of Local-minima using Optimization Algorithms

(a) A drawing by FR without any edge crossing.  
(b) A drawing by FR with an avoidable edge crossing.

Fig. 5.1 Two different drawings with two different initial node placement for the same graph that lead to two different layouts.

greatly influence this outcome and potentially help the algorithm to avoid local-minima. Figures 5.1a and 5.1b illustrate the fact that two different initial layouts can lead to two different final layouts, one of which contains an avoidable edge crossing.

The problem of local-minima does not only happen on the single-level graph drawing algorithms such as Fruchterman and Reingold and Kamada and Kawai but also on the subclass of multilevel graph drawing algorithms. Although the multilevel graph drawing algorithms reduce the chance of local-minima, still there is no guarantee if the generated layout has no crossing even for very small graphs. This is mostly because of the placement step of the multilevel algorithms, in which some algorithms use random placement. Figure 5.2 shows an example of a small graph with 33 nodes and 69 links with 4 different layouts with the multilevel algorithm by Hu [60] implemented in GraphViz [41]. GraphViz is a graph visualization package that implements several different graph drawing algorithms. *sfdp* (Scalable Force Directed Placement) is the implementation of the multilevel algorithm by Hu [60] in GraphViz.

As this figure illustrates, the multilevel algorithms do not guarantee a layout without edge crossing and for every new run the result will be different. A layout without unnecessary crossings is illustrated in Figure 5.3.

Most force-directed algorithms start with a random Initial Node Placement (INP) which has a certain amount of energy. The energy is usually calculated using the summation of attraction and repulsion forces on the graph. As the system has opposite forces (attraction and repulsion) the algorithms ideally try to minimize the total energy of the system in order to bring the system to an equilibrium state. The term energy is sometimes interpreted as stress for some certain algorithms such as the stress-majorization [39] and the goal is to minimize the stress of the function. However calculating the stress in those functions is different from calculating the energy in other
5.1 Introduction

Fig. 5.2 Four different layouts for a graph with 33 nodes and 69 edges by sfdp implemented in GraphViz.

Fig. 5.3 A layout without unnecessary crossings by FR.
algorithms. This concept of energy is used in almost all of the force-directed algorithms, also even in multilevel algorithms [122, 60, 9]. There is always a chance of a large initial amount of energy in the system when the system starts with random Initial Node Placement (INP). Consequently it takes longer to minimize the total energy, however sometimes the system is trapped into a local-minima which even if the system continues for ever it can not get rid of it.

Therefore we can conclude that there are two possible issues which a random Initial Node Placement (INP) can cause: 1) Local-Minima and 2) Long Running Time. However the problem of local-minima is more important because the running time can be less important when powerful machines are employed. Also, for small range graphs the quality is always more important than running time. Not all different types of graphs suffer from a random INP equally. We have found a class of graph topology which suffers more than other types of graphs from a random INP.

5.2 Star-Shaped Graphs

We introduce a subset of un-directed and connected graphs termed star-shaped graphs with certain characteristics based on the betweenness centrality.

Betweenness Centrality

Betweenness centrality is a parameter that indicates how theoretically central a node is. Each pair of nodes in an undirected and connected graph has a theoretical distance or geodesic distance. The theoretical distance is the shortest path between a pair of nodes. The shortest path between a pair of nodes \((n_i, n_j)\) is the number of edges on the shortest path. For instance, the shortest path between node \(n_1\) and \(n_5\) in Figure 5.4 is 3 as there are only 3 edges on the path \((e_1, e_3\) and \(e_4))\). Having the shortest path of a pair of nodes, one can find the nodes that are met along the shortest paths; i.e. the nodes \((n_3\) and \(n_4)\) are met when the shortest path between \((n_1\) and \(n_5)\) is traversed. In the other word we can say that nodes \((n_3\) and \(n_4)\) are between nodes \((n_1\) and \(n_5)\).

The all-pairs shortest paths of a set of nodes \((N)\), has \(\frac{|N| \times (|N| - 1)}{2}\) different shortest paths. The nodes along each shortest path are identified and their frequencies are counted.

The frequency of each node along all-pairs shortest paths is called the betweenness centrality of that node. Figure 5.5 shows the betweenness centrality of all nodes along with the symmetrical matrix which shows the nodes that are met along each shortest path of a pair of nodes (row index and column index).
5.2 Star-Shaped Graphs

Fig. 5.4 A graph with 5 nodes and 4 edges.

Fig. 5.5 A small graph with 5 nodes and 4 edges and the betweenness centrality of all nodes.

\[ \begin{array}{|c|c|c|c|c|}
\hline
 & n_1 & n_2 & n_3 & n_4 & n_5 \\
\hline
n_1 & 0 & n_3 & N/A & n_3 & (n_3, n_4) \\
\hline
n_2 & 0 & N/A & n_3 & (n_3, n_4) & \\
\hline
n_3 & 0 & N/A & n_4 & & \\
\hline
n_4 & 0 & N/A & & & \\
\hline
n_5 & 0 & & & & \\
\hline
\end{array} \]

\( n_1 \): 0 (Betweenness centrality) frequency
\( n_2 \): 0 (Betweenness centrality) frequency
\( n_3 \): 5 (Betweenness centrality) frequency
\( n_4 \): 3 (Betweenness centrality) frequency
\( n_5 \): 0 (Betweenness centrality) frequency
A node with a large betweenness centrality is identified as a more central node in the graph which means a larger number of shortest paths pass through that node.

5.2.1 Star-Shaped Topology

Typically large star-shaped graphs (over 1k nodes) look like a tree with one or a few main root(s) and several different branches with different lengths and weights. They are usually very sparse with long branches. Star-shaped graphs are also similar to scale free networks. A scale free network is a network which the nodes’ degree distribution follows the Power law that means there are a few nodes with large degree of node (see Section 2.1) and many nodes with lower degree of node. Those nodes with large degree are called hubs. Star-shaped graphs are similar to scale free networks in a sense that those hubs do not necessarily have large degree but they have large betweenness centrality. For instance in Figure 5.6 three nodes \( n_a, n_b, n_c \) are circled with different betweenness centrality. Node \( n_a \) has the largest betweenness centrality and then \( n_b \) and then \( n_c \) \( (Bt(n_a) > Bt(n_b) > Bt(n_c)) \). However \( deg(n_b) = 5 \) while \( deg(n_1) = 4 \) and \( deg(n_c) = 4 \).

Typically a star-shaped graph has a few nodes with large betweenness centrality and rest of the nodes with small betweenness centrality. If the betweenness centralities of all nodes are sorted from the largest to the smallest, then the betweenness centrality degree
distribution (BCDD) has a diagram similar to the power law diagram. Figure 5.7a and 5.7b are two examples of star-shaped graphs with their BCDD diagrams that imply the power law. Figure 5.7c which is a $8 \times 8$ mesh, and its BCDD is not similar to the power law diagram. Trees are also categorized as star-shaped graphs and as it is illustrated in Figure 5.7b, the BCDD of that tree follows the power law. Star-shaped graphs happen in some hierarchical networks with more than one main hub while all hubs are connected to each other; e.g. a LAN architecture when a few MAUs (Multi-station Access Unit) are connected to each other and each one services one or two server and each server supports multiple PCs. Another example is in big organizations with several different sections, for instance the network of colleagues in a University with multiple departments and each department has multiple courses and the connections between them if they share any module. Figures 5.8 and 5.9 show two examples of them.

After doing some investigation on star-shaped graphs, we have discovered that star-shaped graphs are one of the vulnerable types of graphs in terms of getting trapped into local-minima easily. Even the new generation of graph drawing algorithms (Multilevel paradigm) which claims to reduce the chances of local-minima can not help this issue even for a very small graphs of less than 50 nodes (see Figure 5.2). In this figure four different layouts for the same graph with 33 nodes and 69 nodes are visualized using the well-known multilevel algorithm by Hu [60] implemented in Graphviz [41]. As we can see three of them are trapped into local-minima. As a very general point of view, star-shaped graphs look like the layout in Figure 5.10 in which it has a few hubs with large betweenness centrality and some heavy branches which contain several nodes inside.

### 5.2.2 How to Generate Star-Shaped Graphs

In order to perform our evaluation we generate several different star-shaped graphs with different sizes. Star-shaped graphs are similar to the scale-free graphs (networks). The graph starts to be generated node by node. After adding each new node the betweenness centrality of all the currently existing nodes are evaluated again. Those nodes that have smaller betweenness centrality have higher probability of being adjacent to the new node that is going to be added into the system. Obviously each node can have more than one link (depending on the average degree of the graph which can be set initially). When a new node makes its first connection (link) to an existing node, then for the next connections (if applied) those nodes with smaller shortest path will be more likely to be connected to the new node as the second or later connection.
Investigation into the Reduction of Local-minima using Optimization Algorithms

Fig. 5.7 Three different graphs with their BCDD diagrams.
Fig. 5.8 A LAN architecture.

Fig. 5.9 University network illustrating the offered modules by an university and their connections if they share any module.
To do this, connections are not being made between new nodes and nodes with large betweenness centrality. This implies the hierarchy systems where there are a few central (important) nodes and each of them has a separate branch. For instance, there will be a few nodes with larger betweenness centrality and some branches with relatively high density.

5.3 INP: Initial Node Placement Effect

As it was mentioned earlier, this class of graphs (star-shaped) is more vulnerable to getting trapped into a local-minimum. A star-shaped graph has a few nodes with high betweenness centrality and several nodes with lower betweenness centrality. This constitutes a graph with a few main hubs (not necessarily high degree) and some relatively heavy branches, (see Figure 5.10). In the class of force-directed algorithms, forces of two different types are imposed on each node by other nodes. Usually there are attraction and repulsion forces which depend on the applied algorithm, forces imposed between all pairs of nodes (repulsion force) or adjacent nodes (attraction force). However there are other algorithms which employ attraction and repulsion forces in other forms such as stress, that is basically the difference between the ideal distance and the physical distance between each pair of nodes. Since there are some heavy masses in star-shaped graphs, sometimes the graph might be located in a situation that two big masses have to pass each other in order to get rid of a major edge crossing but those two big masses impose repulsion forces to each other and do not let each other to move in order to pass each other. Figure 5.11 shows a star-shaped graph which two masses (M1) and (M2) are trapped as they impose repulsion forces to each while they are trying to pass each other in order to get rid of the edge crossing. All the nodes in (M1) impose repulsion force to all the nodes in (M2) which makes a large repulsion
force between (M1) and (M2) and does not let them to take the ideal move in the
direction of green and orange arrows (Figure 5.11).

The problem illustrated in Figure 5.11 can arise when a random INP in any iterative
schema graph drawing algorithm [38, 70, 82, 39, 30] is involved. Even those algorithms
that apply the multilevel paradigm [122, 60, 9], can not guarantee a layout without
local-minima, (see Figure 5.2). Although multilevel-based algorithms do not start the
drawing process with all of the individual nodes initially (only one or a few coarsest
node(s)), at the beginning of each level the coarser nodes are broken down and new
nodes are subjected to a new placement. The Placement phase in multilevel algorithms
is most of the times performed randomly.

One of the reasons that multilevel algorithms are employed is to reduce the running
time for large graphs i.e. over 500 nodes. This is because multilevel algorithms
optimize the complexity, however the quality might not be as good as single-level
algorithms. Therefore for small graphs less than 500 nodes, single-level algorithms
such as Fruchterman and Reingold or Kamada and Kawai, or stress majorization, can
be more beneficial; Figure 5.12 shows two layouts for a mesh graph 8 × 8 with 64
nodes. The layout in Figure 5.12a is drawn using a multilevel algorithm [60] and the
layout in Figure 5.12b is drawn using stress majorization technique that is a single-level
algorithm. The layouts are drawn by Graphviz [41].

Although the single-level algorithms tend to produce fairly good layouts, they still
suffer from a random initial node placement. Figure 5.13 shows three layouts for a
star-shaped graph with three well-known algorithm a) Stress majorization b) The
multilevel approach c) Fruchterman and Reingold. A local-minimum (edge crossing)
occur in all these three drawings, and this is because of the random node placement
in both single-level and multilevel algorithms.

To address the problem mentioned above, we are going to present a general solution
along with some results for single-level algorithms to address the local-minima issue.
Investigation into the Reduction of Local-minima using Optimization Algorithms

Fig. 5.12 Two drawings on the same graph with 64 nodes with two different techniques.

Fig. 5.13 Three drawings on the same graph with 39 nodes with three different techniques.
This solution is mostly for fairly small and average size graphs or those graphs that can be drawn using single-level approaches. This solution focuses on designing the first iteration of the graph drawing algorithms instead of having a random, one in order to reduce the chance of getting trapped into a local-minimum.

Hu et al. [63] proposed a solution for small graphs as well. In that work a node \((v_k)\) that causes the most edge crossings is found from the random initial placement and that node is moved to the center of drawing. Then the nodes adjacent to the initial node \((v_k)\) are evenly distributed around a circle centered at \((v_k)\). This process continues until all the nodes are placed. Although this method is designed not only for trees, but also for other type of graphs this may not produce the best solution; there are better and computationally less expensive approaches to solve this problem for trees.

For instance, our work on Angular Tree Drawing (ATD) at [114] shows a linear approach to drawing any tree without using any force-directed algorithm to produce layouts without any edge crossings. Our proposed method for trees can either be a stand-alone tree drawing approach or it can be used as a pre-processing approach for any other iterative-schema tree drawing. Figure 5.14 shows the layout of a tree with 2000 nodes which is drawn by our tree drawing algorithms. In that work we introduced two slightly different algorithms in order to draw a tree (ATD1 and ATD2). The centric or bi-centric root(s) is/are found and placed at the center of the drawing, then the area around those nodes are divided evenly or according to their degree to accommodate further nodes. One of the algorithm (ATD1) reserves spaces for possible future nodes so then there is no need to re-calculate the layout (see Figure 5.14a) and the other algorithm (ATD2) uses all the spaces to spread nodes around (see Figure 5.14b). The running time for drawing a tree with 2000 nodes is 0.2 sec with either algorithms in Figure 5.14 (a, b). As mentioned earlier, these two algorithms can act as stand-alone tree drawings or they can be the INP for any iterative-schema tree drawing algorithms. These two algorithms guarantee a layout without any single edge crossing therefore there is a large probability that the force-directed algorithm also ends up without any edge-crossing.

Our other solution to this problem deals with more general or complicated graphs and not only trees. The problem of designing an initial node placement is considered as a NP-hard problem as it can be reduced to the TSP (traveling salesman problem). Our solution is presented by two well-known techniques Genetic Algorithms (GA) and Simulated Annealing (SA). Although GA and SA have more general applications in the field of graph drawing [15, 33, 27, 51], we use GA and SA in order to draw
Investigation into the Reduction of Local-minima using Optimization Algorithms

A drawing by ATD1 algorithm

A drawing by ATD2 algorithm

Fig. 5.14 A tree with 2000 nodes using ATD algorithms.

The Genetic Algorithm (GA) was originally developed by Holland [58]. The GA is a meta-heuristic search algorithm that simulates the natural phenomena inspired by Darwin’s theory [31]. Genetic algorithms operate to improve the quality of a population and are a subset of the more general class of evolutionary algorithms. GAs have application in many different disciplines such as bioinformatics, engineering, chemistry, economics, mathematics and also geometry. They operate on strings, inspired by chromosomes in natural organism, that can carry different types of values such as numerical, binary or even character. A GA normally starts with a certain number of random orderings (chromosomes) called a population. The top orderings from the population are selected for getting mutated and modified for the next generation. In our case, each ordering is a suggested design for the initial node placement of a graph drawing algorithm. The GA starts with some randomly generated orderings that
represent the initial node placement to be given to any force-directed and iterative graph drawing algorithm.

### 5.4.1 GA Techniques

The genetic algorithm is an iterative process which starts from an initial random population and over a number of iterations new generations are produced and in general the fitness cost of the new generations’ orderings converge towards the solution. There are a few techniques at each iteration which are applied on the whole population such as selection, crossover, mutation and the fitness function.

#### Fitness Function

The fitness function takes an ordering and tells us how good that ordering is (fitness cost). The fitness function has to be defined based on the given problem. For each GA the fitness function should be defined based on the problem’s type. There are a number of different types of crossover, mutation and selection techniques [83] defined already which they can be employed in the given problem directly or with slight changes.

#### Selection

The selection technique is a process of selecting the top orderings from a given population. The selection technique is performed based on the fitness cost of the orderings.

#### Tournament Selection

Tournament selection is a procedure that filters the low quality orderings and makes copies of high quality orderings. There is a parameter which is called the tournament size $t$. The tournament is applied on the given population (generation) $p$ times where $p$ is the size of the population. At each tournament round, $t$ different orderings are randomly selected. Out of $t$ randomly selected orderings, the best one (based on the fitness cost) is selected and added into the filtered population list. This process repeats $p$ times until the filtered population is filled in with relatively improved orderings. The tournament size is usually very much smaller than the size of the population for reasons of efficiency.
Elitism

Elitism is another selection procedure which guarantees the progression of high ranking individuals from one generation to the next. In our implementation all orderings in the population are sorted from the best towards the worst based on their fitness cost. Then the population is divided into 3 different sections, $s_1$, $s_2$ and $s_3$. $s_1$ has the best orderings $s_2$ has the average orderings and $s_3$ has the worst. At this point the last section $s_3$ is replaced by $s_1$ so then there are two copies of the best orderings and one copy of the average ordering. One of the copies of the best orderings is sent directly to the next generation. Therefore elitism guarantees that the fitness cost of generations never goes down.

Crossover

Crossover is another GA technique in which two orderings of a given population are mixed in some fashion and produce one or two new ordering(s). The details of the specific crossover operators that are used in this work are given in the subsequent sections.

Mutation

Mutation is another GA technique that simply takes one ordering and makes a slight change to it in order to generate a new ordering.

Reproduction

To implement the genetic algorithm we need our population to breed and consequently create a new generation. This can not be obtained unless individuals (orderings) in the population get mixed or get permuted to breed novel generation. But this should not be applied on all individuals (orderings) in population as there is no guarantee of producing a better generation after mixing all the individuals. Therefore some individuals should be allowed to simply reproduce themselves for the next generation. This is called reproduction. By using this technique we keep the specification of the previous generation on into the next generation while we have novel children by making use of crossover and mutation.
5.4.2 GA Initiation

A genetic algorithm generally starts with a randomly generated population \((p\) number of potential solutions) where \(p\) is the size of the population. The randomly generated population is also sometimes called generation \(0\). The initial population then undergoes by the four main techniques: selection, crossover, mutation and reproduction in order to create the new generation. This process continues for a certain number of generations, then the best ordering from the last generation is generally picked in order to be the result of GA.

5.5 Dist-GA: Distance Based Graph Drawing Initiation using GA

Positioning the nodes on a Cartesian space (2D or 3D) with regards to their (geodesic) theoretical distances improves the aesthetic qualities of the layout \([78]\). Our experiments have shown that if a force-directed graph drawing algorithm is initialized using a smart node placement using geodesic distance instead of random nodes placement, then the chance of edge crossings is significantly reduced. The first question that arises is what type of environment is needed. For the sake of simplicity we have used a one dimensional environment or a linear layout. For instance a small graph with 6 nodes is illustrated in Figure 5.15; part (a) shows the general structure of the graph and parts (b) and (c) show two different orderings with their corresponding one dimensional layouts.

Although working on 1-dimensional space is computationally less expensive, this initial node placement does not allow the force-directed algorithm to run properly. Our solution was to bend the straight line to create an arc or even to create a complete circle. Doing so, we benefit from the lower computational processing needs while a 2 dimensional layout is generated as the initial node placement of a graph drawing. Our initial results on using theoretical distances in order to produce a smart initial node placement were published in Gecco 2015 \([116]\).

5.5.1 Fitness Function

The idea of this work is to use discrete positions for each node on a 1-dimensional space as a straight line but modifying that line into a curve or a complete circle. The length of the line is initially divided by \(|N|−1\) where \(|N|\) is the number of nodes. Starting
Investigation into the Reduction of Local-minima using Optimization Algorithms

(a) A layout for a graph with 6 nodes.

(b) An ordering of nodes for a graph with 6 nodes.

(c) An ordering of nodes for a graph with 6 nodes that causes a few edge crossings.

Fig. 5.15 (a) shows the general structure of a graph with 6 nodes, (b) shows an example of an ordering with its layout in 1-dimensional space, (c) shows another ordering with its layout in 1-dimensional space.
5.5 Dist-GA: Distance Based Graph Drawing Initiation using GA

From one of the heads of the line, nodes are accommodated. Therefore an ordering of nodes is created by picking the index of the nodes from the head or tail of the line. The line then is bent and the goodness of that ordering is calculated based on the geodesic distances of nodes in the graph. The calculation is simple and it is as follows:

$$\text{Cost} = \left( \sum_{i=0}^{[N]} \sum_{j=0}^{[N]} |SP(n_i, n_j) - (|IN_{n_i} - IN_{n_j}|)| \right)$$

where $SP(n_i, n_j)$ is the theoretical distance or shortest path between nodes $n_i$ and $n_j$ and $IN_{n_i}$ is the index of $n_i$ in the given ordering. An ordering with a lower cost (fitness cost) is a better ordering and can be a better choice for the graph drawing algorithm. Figure 5.16 shows two different orderings accommodated on an arc (half circle) with two different fitness costs. The fitness cost of the ordering that is shown in Figure 5.16(A) is equal to 4 and the fitness cost of the other ordering in Figure 5.16(B) is equal to 22.

A smart initial node placement for the force-directed algorithm not only helps to reduce the chance of local-minima but it can also reduce the number of iterations. As it has been mentioned earlier, this fitness function initially needs to calculate the all-pairs-shortest-path in order to calculate the fitness cost. With a larger graph, the algorithm needs to spend more time in order to find all-pairs-shortest-path.

5.5.2 Dist-GA Techniques

When the filtered population is filled in by elitism selection then some of the orderings are sent for possible modifications. GA probabilistically selects one of the techniques (crossover, mutation or reproduction) in order to apply it on the selected orderings. Usually the largest probability is assigned to reproduction and then crossover and...
Investigation into the Reduction of Local-minima using Optimization Algorithms

Crossover

Crossover simply takes two orderings (ord1 and ord2) and randomly selects a cutting point $cp$ in the range of $[2, n-2]$ where $n$ is the size of each ordering or the number of nodes. $cp$ then creates two sub-orderings for each ordering as follows: $ord1.1 = [1, cp]$, $ord1.2 = [cp+1, n]$, $ord2.1 = [1, cp]$, $ord2.2 = [cp+1, n]$. Then the second sub-ordering of the first ordering is replaced by the second sub-ordering of the second ordering. (see Figure 5.17). After replacement there might be a few repeated nodes in one ordering and a few missing nodes. Those repeated and missed nodes are found and get swapped.

5.6 Ang-GA: Angular Based Graph Drawing INITIATION using GA

In this section we propose a genetic algorithm with a different fitness function compared with Dist-GA proposed in Section 5.5. This GA is more useful for star-shaped graphs. As it was mentioned earlier, star-shaped graphs are more likely to be trapped into local-minima, the reason for this is visually depicted in Figure 5.11. This new fitness function focuses more on the branches and their directions towards each other instead
Fig. 5.18 The general structure of our proposed GA.
Investigation into the Reduction of Local-minima using Optimization Algorithms

of the shortest path between pairs of nodes. Our results using this proposed Genetic algorithm have been published in Gecco 2016 [117].

5.6.1 Fitness Function

The GA aims at maximizing the fitness cost which is generated using a weighted sum of angles between adjacent edges. For each pair of adjacent edges, we consider the smaller angle enclosed by these edges. This is similar to the approach of Branke et al. whose fitness function is based on the average angle between adjacent edges [16].

The fitness of a layout $\gamma$ for a graph $G = (N, E)$ with a set of nodes $N = \{n_1, n_2, \ldots, n_{|N|}\}$ and a set of edges $E \subseteq N \times N$ is expressed in Equation (5.1) where $a(n_i, n_j, n_k)$ is the angle between two line segments $l_{ij} = \overline{p_{n_i}p_{n_j}}$ and $l_{ik} = \overline{p_{n_i}p_{n_k}}$ and the weights $Bt_{n_i}, Bt_{n_j}$ and $Bt_{n_k}$ are the betweenness centralities of nodes $n_i, n_j$ and $n_k$, respectively; $|l_{ij}|$ are $|l_{ik}|$ are the lengths of the two line segments created by the positions of those three nodes in the layout $\gamma$. Bannister et al. [4] also made use of the betweenness centralities of nodes in order to keep a vertex with a relatively high betweenness centrality not too far from the center of the layout. In this work, betweenness centrality is used in the context of initial node placement for any force-directed algorithm.

$$f(\gamma) = \sum_{n_i \in N} \sum_{n_j, n_k \in N, \{n_i, n_j\} \in E, \{n_i, n_k\} \in E} a(n_i, n_j, n_k)Bt(n_i)Bt(n_j)Bt(n_k)|l_{ij}|l_{ik}|$$  \hspace{1cm} (5.1)

By using this fitness function, angles at nodes with higher betweenness centrality and longer links are considered more important when evaluating the fitness of a layout. By doing this we aim at reserving appropriate and sufficient space for nodes that are connected to the nodes with higher betweenness centrality. We consider angles at nodes with low betweenness centrality less important when evaluating the fitness. This is because these nodes are theoretically less connected to the rest of the graph and it should be easier to move in order to find an aesthetically pleasing placement by a force-directed algorithm. After doing extensive initial experiments we found that we can further refine the fitness function by excluding angles between some edges. We found that maximizing the angles between edges in 3-cliques (triangles) has no significant effect on the results; therefore they can be excluded. Furthermore, in the case when an angle is formed between edges $\{n_i, n_j\}$ and $\{n_i, n_k\}$ where nodes $n_j$ and $n_k$ have the same betweenness centrality and lower than the one of $n_i$, then the layout does not benefit from maximizing this angle.
5.6 Ang-GA: Angular Based Graph Drawing Initiation using GA

Fig. 5.19 Two graphs with two different layouts.

In Figure 5.19 two graphs are shown with two different layouts for each. Figure 5.19a and Figure 5.19c show improved layouts in terms of edge crossings than Figure 5.19b and Figure 5.19d. In Figure 5.19a and Figure 5.19b, the triangle and circle nodes have higher betweenness centrality than the diamond and square nodes, therefore the bigger the angles created by the nodes with higher betweenness centrality the better the fitness value (see Equation 5.1). The same scenario is shown in Figure 5.19c and Figure 5.19d where the triangle and circle nodes have higher betweenness centrality than the diamond and square nodes. The graph in Figure 5.19c and Figure 5.19d has been taken from the Rome Graphs dataset [10].

5.6.2 Ang-GA Techniques

Selection

We use an elitist technique as described in Section 5.4.1 for selecting individuals to be preserved in the next generation.

Crossover

The crossover operator takes two individuals (parents) and combines them in order to produce two new individuals (children) for the next generation. Various crossover
operators have been proposed in the research literature, such as position based crossover (POS), alternating position crossover (AP), partially mapped crossover (PMX) and Voting Recombination Crossover (VR) [83]. Since our individual represents a set of interconnected nodes we wish to keep a good part of the layout preserved when performing crossover.

After performing some experiments we have decided to choose PMX as our crossover operation. It randomly selects $n_i$ such that $n_i \neq n_{|N|}$ and $n_i \neq n_1$. Then it creates the first child (or offspring) from the set of $\{n_1...n_i\}$ from the first parent and the set of $\{n_{i+1}...n_{|N|}\}$ from the second parent and creates the second child from the set of $\{n_1...n_i\}$ from the second parent and the set of $\{n_{i+1}...n_{|N|}\}$ from the first parent. Our PMX crossover is illustrated in Figure 5.20; $n_4$ is chosen and then the set $\{n_1, n_2, n_3, n_4\}$ from the first parent and the set $\{n_5, n_6, n_7\}$ from the second parent create the first child, while the complementing vertex sets from each parent create the second child.

**Mutation**

The mutation operator we chose to use in our GA is *SingleMutate*, introduced by Groves et al. [51] and also used in TimGA [33]. The reason behind our choice is that we want our mutation to change the positions of very few nodes in the layout, thus being less intrusive than our crossover operator. This way we have a crossover operator that can produce new individuals which are very different from their parents and a mutation operator that creates relatively small changes. Our mutation randomly picks
Fig. 5.21 Mutation (SingleMutate) randomly picks two nodes from the layout in the left and moves them into two randomly picked empty cells. The layout on the right is the result of this mutation.

two different nodes and moves them to two, also randomly picked, empty cells in the drawing matrix (see Figure 5.21).

5.6.3 Functionality

As mentioned earlier, this GA only creates the INP for a force-directed graph drawing. A visual result of this GA for a graph with 39 nodes and 45 edges with its evolution under the Fruchterman and Reingold algorithm is shown in Figure 5.22. The first layout surrounded by a black border is the result of Ang-GA. Although the initial layout is not very clear, it accommodates the main branches in a way to avoid the problem discussed in Figure 5.11 and this is useful for star-shaped graphs with heavy branches.

5.7 Simulated Annealing

Simulated Annealing usually starts with a randomly generated candidate as a solution or an ordering. Simulated annealing introduces a new term which is called temperature. Typically the temperature is the amount of modification that is applied on an ordering. The temperature of the system is initially high, therefore the candidate is allowed to modify itself by a high rate (proportional to the temperature) with the hope of finding a better solution. As the temperature gets lower (system is getting cooler) the candidate modifies itself with lower rate until the system is, so called, frozen, and no more modification is allowed. The operation which does the modification on the candidate is called mutation analogous to the mutation operator in GAs. The mutation and the fitness function of our proposed simulated annealing are similar to our genetic algorithm represented in Section 5.5.
A candidate (ordering) in our algorithm represents a sequence of node indices similar to the genetic algorithm. We consider a discrete drawing space and represent a layout by a circle which can be easily converted to a linear environment for the sake of computational simplicity (see Figure 5.23). The considered circle has a radius equal to 1 and its border is divided by $|N|$ where $|N|$ is the number of nodes. Two different orderings for the same graph with 9 nodes are shown in Figure 5.24. Figures 5.24a and 5.24b show how two different orderings $[5-3-2-1-4-7-8-9-6]$, generated by simulated annealing, and $[2-1-5-7-9-6-8-4-3]$, generated randomly, on a circle create two different initial vertex placements according to their orderings. Figure 5.24c represents the final placement (layout) by Fruchterman and Reingold algorithm [38].

5.8 Results

In this section the results of applying the three aforementioned techniques in order to produce a smart INP are presented. As mentioned earlier, SINP is just a pre-processing step for any force-directed and iterative schema graph drawing algorithm. This means that SINP is not involved in drawing the graph but only used to prepare a better initial node placement. To evaluate the efficiency of applying the proposed pre-processing step on a force-directed algorithm one needs to check for the local-minima and one of the fastest way to check local-minima is to count the number of edge-crossings. To do
5.8 Results

Fig. 5.23 One example of an ordering on a circle with their polar coordinates, e.g. the distance between the nodes 3 and 5 is the same as the distance between the nodes 2 and 8 and it is equal to 1.41.

Fig. 5.24 A graph with 9 vertices with two different orderings with their corresponding initial vertex placements (5.24a and 5.24b) respectively. The actual layout using the Fruchterman and Reingold algorithm is shown in Figure 5.24c.
so we run the drawing two times, first, with a random INP, and secondly with a smart INP (the proposed pre-processing step). From now on a drawing-run that starts with a random INP is called the R-drawing and a drawing-run that starts with a smart INP is called the S-drawing. The number of edge crossings at each run is counted. If the R-drawing has less edge crossings than the S-drawing then S-drawing fails and if the S-drawing has less edge crossings than the R-drawing then the S-drawing wins. In this work we exploit three different graph typologies in our evaluation using the algorithm by Fruchterman and Reingold [38]:

1. Star-Shaped Graph: (see Section 5.2).

2. Cactus Graph: Cactus graph is a special type of graph such that no two cycles share an edge, a tree can be a cactus graph [126].

3. Random Graph: The third data set is just a random graph that is created with a graph density around $0.1$.

The total number of edge-crossings with both types of drawings (the R-drawing and the S-drawing) is measured. However the total number of edge-crossings is not a very precise factor for our comparison. This is because for non planar graphs there is always a minimum number of edge crossings $e$. If the S-drawing produces a layout with $e$ edge crossings and the R-drawing produces a layout with $e + 1$ edge crossings then the difference would not statistically seem significant especially if $e$ is large, while the extra edge crossing by the R-drawing might change the quality of the layout at a noticeable level. Figure 5.25 shows two drawings; one with 6 edge crossings (see Figure 5.25a), and the other one with 7 edge crossings (see Figure 5.25b). The extra edge crossing changes the quality of the layout by a large degree. Therefore the total number of edge crossings may not be always a good factor if a complete force directed algorithm is performed. Our experiments perform two runs for each test (the R-drawing and the S-drawing) and scores the one with the lower crossings as the winner.

To reduce the error due to the random initiation, we performed the experiment several different times and took the average of these experiments.

5.8.1 p-value

In order to show the usability of the proposed techniques (GAs and SA) we make use of a well known statistical hypothesis test called the $p$-value.

Let $A$ and $B$ be the two lists of numerical results obtained by two algorithms $Alg.A$ and $Alg.B$ respectively on the same type of data and solving the same problem. The
Fig. 5.25 Two layouts with only one edge crossings difference.

(a) A layout of a graph with 27 nodes with 6 edge crossings.  
(b) Another layout of a graph with 27 nodes with 7 edge crossings.

p-value is the probability for Alg.B to obtaining a list of numerical results (B) very similar to the results of Alg.A (A) [37]. A large p-value indicates a large similarity between the results by Alg.A and Alg.B and a small p-value indicates a more different results by Alg.A and Alg.B. Most of the studies make use of the threshold at 0.05 as the cutoff for significance [87]. If the p-value is less than 0.05, then the difference between two algorithms Alg.A and Alg.B are considered as significant.

5.8.2 Evaluation

The main experiment by the algorithms proposed in this chapter is divided into three categories according to the graph topology (St for star-shaped, Ca for cactus and Ra for random graphs). Each category is tested with \( m \) different graphs in terms of their sizes starting from node-count (size) 50; e.g. St.50, St.51...St.90, Ca.50, Ca.51...Ca.90 and Ra.50, Ra.51,...Ra.75. 50 different graphs are generated for each node-count. Finally each graph is tested 50 times. Therefore, each node-count in each category is tested 2500 times. Each single test contains two runs (the S-drawing and the R-drawing). The number of edge crossings for each run at each test is counted and the winner of each test is identified. The number of wins for each graph, out of 50 different tests, is counted for both S-drawing and R-drawing. The average of wins for each node-count is calculated again for both the S-drawing and the R-drawing. Then each run (the S-drawing and R-drawing) has a value in the range of \([0, 50]\). This value shows the number of wins for each drawing (S-drawing and R-drawing). Note: The sum of this value for S-drawing and R-drawing is not necessarily 50 because there might be some cases that S-drawing and R-drawing are producing layouts with an equal number of edge crossings. For each category two lists \( l_R \) and \( l_S \) for the R-drawing and the S-drawing are presented. Each list contains \( m \) values where \( m \) is the number of different node-counts, e.g. St.50...St.90 so \( m = 41 \); each value represents the number of wins.
To measure if the number of wins by the S-drawing is significantly greater than the R-drawing the p-value of the two lists \( l_R \) and \( l_S \) is calculated. The average running time for each single run is also calculated. To represent the results easier we generate the average of each group of 10 different node-counts; e.g. \( St.70, St.71, ..., St.79 \) and represent them in one row. For example a row in a table as follows:

<table>
<thead>
<tr>
<th>( N )</th>
<th>( E )</th>
<th>( R - Crs )</th>
<th>( S - Crs )</th>
<th>( R.wins )</th>
<th>( S.wins )</th>
<th>( R.time )</th>
<th>( S.time )</th>
<th>( p - value )</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>80.91</td>
<td>2.404</td>
<td>1.914</td>
<td>15.875</td>
<td>21.845</td>
<td>0.037</td>
<td>0.59</td>
<td>0.0000206</td>
</tr>
<tr>
<td>65</td>
<td>70.445</td>
<td>2.100</td>
<td>1.629</td>
<td>14.915</td>
<td>20.935</td>
<td>0.025</td>
<td>0.403</td>
<td>0.0000241986</td>
</tr>
<tr>
<td>75</td>
<td>80.911</td>
<td>2.404</td>
<td>1.914</td>
<td>15.875</td>
<td>21.845</td>
<td>0.037</td>
<td>0.59</td>
<td>0.0000206103</td>
</tr>
<tr>
<td>85</td>
<td>90.933</td>
<td>2.650</td>
<td>2.172</td>
<td>16.907</td>
<td>22.364</td>
<td>0.053</td>
<td>0.83</td>
<td>0.000239976</td>
</tr>
</tbody>
</table>

means that the average node-count is 75, the average number of edges is 80.91, the average number of edge crossings per single run when the R-drawing is applied is 2.404 \( (R - Crs) \) and 1.914 when the S-drawing \( (S - Crs) \) is applied. The average number of wins for the R-drawing \( (R.wins) \) and the S-drawing \( (S.wins) \) per 50 tests are 15.875 and 21.845 respectively. The average running time for each single run for the R-drawing \( (R.time) \) and the S-drawing \( (S.time) \) are 0.037 and 0.59 respectively. And finally 0.0000206 shows the p-value of two lists \( l_R \) and \( l_S \) for two runs (the S-drawing and the R-drawing) before making the average of node-counts.

In the following we report our results for each technique in Sections 5.5, 5.6 and 5.7 separately with different tables.

### 5.8.3 Dist-GA

Three different experiments based on three different graph typologies, as mentioned above, are performed with the Dist-GA algorithm presented in Section 5.5. The results of the first experiment on star-shaped graphs are presented in Table 5.1. The results of the second and the third experiments on cactus graphs and random graphs are presented in Table 5.2 and the Table 5.3 respectively.

Table 5.1 The results of applying Dist-GA on star-shaped graphs for node-counts [50, 90].

<table>
<thead>
<tr>
<th>( N )</th>
<th>( E )</th>
<th>( R - Crs )</th>
<th>( S - Crs )</th>
<th>( R.wins )</th>
<th>( S.wins )</th>
<th>( R.time )</th>
<th>( S.time )</th>
<th>( p - value )</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>60.115</td>
<td>1.875</td>
<td>1.452</td>
<td>14.12</td>
<td>20.158</td>
<td>0.016</td>
<td>0.262</td>
<td>0.000430276</td>
</tr>
<tr>
<td>65</td>
<td>70.445</td>
<td>2.100</td>
<td>1.629</td>
<td>14.915</td>
<td>20.935</td>
<td>0.025</td>
<td>0.403</td>
<td>0.0000241986</td>
</tr>
<tr>
<td>75</td>
<td>80.911</td>
<td>2.404</td>
<td>1.914</td>
<td>15.875</td>
<td>21.845</td>
<td>0.037</td>
<td>0.59</td>
<td>0.0000206103</td>
</tr>
<tr>
<td>85</td>
<td>90.933</td>
<td>2.650</td>
<td>2.172</td>
<td>16.907</td>
<td>22.364</td>
<td>0.053</td>
<td>0.83</td>
<td>0.000239976</td>
</tr>
</tbody>
</table>
Table 5.2 The results of applying Dist-GA on cactus graphs for node-counts [50, 90].

| $|N|$ | $|E|$ | $R - Crs$ | $S - Crs$ | $R.wins$ | $S.wins$ | $R.time$ | $S.time$ | $p - value$ |
|-----|-----|-------|-------|--------|--------|--------|--------|----------|
| 55  | 62.153 | 1.084 | 0.641 | 9.087 | 22.153 | 0.016 | 0.262 | 2.67373E-15 |
| 65  | 73.293 | 1.445 | 0.896 | 10.653 | 24.313 | 0.025 | 0.404 | 6.46775E-21 |
| 75  | 84.456 | 1.831 | 1.195 | 12.207 | 25.167 | 0.039 | 0.59 | 2.51503E-20 |
| 85  | 95.367 | 2.261 | 1.54 | 13.235 | 26.013 | 0.054 | 0.829 | 6.76124E-24 |

Table 5.3 The results of applying Dist-GA on random graphs for node-counts [50, 75].

| $|N|$ | $|E|$ | $R - Crs$ | $S - Crs$ | $R.wins$ | $S.wins$ | $R.time$ | $S.time$ | $p - value$ |
|-----|-----|-------|-------|--------|--------|--------|--------|----------|
| 55  | 158.651 | 50.147 | 49.259 | 21.973 | 25.396 | 0.016 | 0.262 | 0.000150446 |
| 62  | 180.139 | 66.534 | 65.549 | 22.369 | 25.392 | 0.022 | 0.366 | 0.012798978 |
| 72  | 210.648 | 91.001 | 89.86 | 23.032 | 25.232 | 0.033 | 0.528 | 0.05570106 |

5.8.4 Ang-GA

Another set of experiments based on three different graph typologies are performed by the Ang-GA algorithm presented in Section 5.6. The first, second and the third experiments are performed in turn on star-shaped, cactus and random graphs and the results are presented in Tables 5.4, 5.5 and 5.6 respectively.

Table 5.4 The results of applying Ang-GA on star-shaped graphs for node-counts [50, 90].

| $|N|$ | $|E|$ | $R - Crs$ | $S - Crs$ | $R.wins$ | $S.wins$ | $R.time$ | $S.time$ | $p - value$ |
|-----|-----|-------|-------|--------|--------|--------|--------|----------|
| 55  | 60.189 | 1.788 | 1.437 | 14.442 | 19.651 | 0.015 | 0.658 | 0.0000289213 |
| 65  | 70.389 | 2.078 | 1.665 | 15.062 | 20.842 | 0.024 | 0.967 | 0.0000883537 |
| 75  | 80.787 | 2.326 | 1.93 | 16.444 | 21.262 | 0.037 | 1.361 | 0.0000307967 |
| 85  | 91.025 | 2.752 | 2.289 | 17.16 | 22.291 | 0.053 | 1.827 | 0.0000724237 |

Table 5.5 The results of applying Ang-GA on cactus graphs for node-counts [50, 90].

| $|N|$ | $|E|$ | $R - Crs$ | $S - Crs$ | $R.wins$ | $S.wins$ | $R.time$ | $S.time$ | $p - value$ |
|-----|-----|-------|-------|--------|--------|--------|--------|----------|
| 55  | 62.184 | 1.112 | 0.753 | 10.502 | 20.978 | 0.016 | 0.571 | 0.0000000482185 |
| 65  | 73.309 | 1.406 | 0.972 | 11.698 | 22.798 | 0.025 | 0.854 | 0.0000000121989 |
| 75  | 84.335 | 1.808 | 1.258 | 12.935 | 24.345 | 0.04 | 1.203 | 0.000000429996 |
| 85  | 95.191 | 2.223 | 1.56 | 13.644 | 25.535 | 0.057 | 1.637 | 0.000000306618 |
Investigation into the Reduction of Local-minima using Optimization Algorithms

Table 5.6 The results of applying Ang-GA on random graphs for node-counts [50, 75].

<table>
<thead>
<tr>
<th>N</th>
<th></th>
<th>E</th>
<th></th>
<th>R − Crs</th>
<th>S − Crs</th>
<th>R.wins</th>
<th>S.wins</th>
<th>R.time</th>
<th>S.time</th>
<th>p − value</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>158.84</td>
<td>49.666</td>
<td>48.946</td>
<td>22.511</td>
<td>24.958</td>
<td>0.016</td>
<td>1.549</td>
<td>0.023980217</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>189.225</td>
<td>72.031</td>
<td>71.129</td>
<td>22.618</td>
<td>25.375</td>
<td>0.025</td>
<td>2.331</td>
<td>0.046671719</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>210.216</td>
<td>89.548</td>
<td>88.502</td>
<td>22.964</td>
<td>25.432</td>
<td>0.033</td>
<td>2.981</td>
<td>0.183198535</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.8.5 SA using Theoretical Distance

Simulated Annealing (SA) as another optimization technique is our last set of experiments. This SA uses a similar fitness function as the Dist-GA uses in Section 5.5. Three different experiments based on three different graph typologies are performed by the proposed SA in Section 5.7. Star-shaped, cactus and random graphs are used for the three experiments and their results are represented in Tables 5.7, 5.8 and 5.9 respectively.

Table 5.7 The results of applying SA on star-shaped graphs for node-counts [50, 90].

<table>
<thead>
<tr>
<th>N</th>
<th></th>
<th>E</th>
<th></th>
<th>R − Crs</th>
<th>S − Crs</th>
<th>R.wins</th>
<th>S.wins</th>
<th>R.time</th>
<th>S.time</th>
<th>p − value</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>60.1</td>
<td>1.881</td>
<td>1.533</td>
<td>14.827</td>
<td>19.867</td>
<td>0.015</td>
<td>0.041</td>
<td>0.000494734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>70.262</td>
<td>2.027</td>
<td>1.695</td>
<td>15.776</td>
<td>20.435</td>
<td>0.024</td>
<td>0.073</td>
<td>0.000403794</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>80.807</td>
<td>2.416</td>
<td>2.076</td>
<td>17.08</td>
<td>21.382</td>
<td>0.037</td>
<td>0.119</td>
<td>0.000130756</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>91.082</td>
<td>2.842</td>
<td>2.418</td>
<td>17.605</td>
<td>22.322</td>
<td>0.053</td>
<td>0.185</td>
<td>0.0000256995</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8 The results of applying SA on cactus graphs for node-counts [50, 90].

<table>
<thead>
<tr>
<th>N</th>
<th></th>
<th>E</th>
<th></th>
<th>R − Crs</th>
<th>S − Crs</th>
<th>R.wins</th>
<th>S.wins</th>
<th>R.time</th>
<th>S.time</th>
<th>p − value</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>62.244</td>
<td>1.152</td>
<td>0.998</td>
<td>13.596</td>
<td>17.633</td>
<td>0.015</td>
<td>0.041</td>
<td>0.001858424</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>73.216</td>
<td>1.445</td>
<td>1.234</td>
<td>14.796</td>
<td>19.742</td>
<td>0.025</td>
<td>0.072</td>
<td>0.000025235</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>84.321</td>
<td>1.808</td>
<td>1.559</td>
<td>15.885</td>
<td>21.14</td>
<td>0.037</td>
<td>0.119</td>
<td>0.00002107</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>95.313</td>
<td>2.190</td>
<td>1.893</td>
<td>16.685</td>
<td>21.955</td>
<td>0.054</td>
<td>0.184</td>
<td>0.00000159525</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.8 Results

Table 5.9 The results of applying SA on random graphs for node-counts [50, 75].

| \(|N|\) | \(|E|\) | \(R - Crs\) | \(S - Crs\) | \(R.wins\) | \(S.wins\) | \(R.time\) | \(S.time\) | \(p - value\) |
|---|---|---|---|---|---|---|---|---|
| 55 | 159.156 | 50.455 | 49.661 | 22.3 | 25.135 | 0.015 | 0.041 | 0.024899708 |
| 65 | 189.491 | 72.931 | 71.972 | 22.533 | 25.287 | 0.024 | 0.073 | 0.074198437 |
| 72 | 209.976 | 91.219 | 90.085 | 22.932 | 25.308 | 0.033 | 0.104 | 0.006871841 |

5.8.6 Statistical Results

An evolutionary algorithm such as a genetic algorithm has several different parameters, e.g. population size, number of generations, GA techniques, probabilities, and so on. By manipulating them one can get slightly different results. Our focus in this work is not on creating a new genetic algorithm or simulated annealing algorithm, our aim is to show that evolutionary algorithms can be of help for relatively small graphs in order to reduce the chance of edge crossings when they are visualized by any iterative force-directed algorithms including our proposed algorithms in Chapters 3 and 4.

Out of three different optimization algorithms presented in this work the total average p-value of each algorithm for each different graph topology along with their average running times are calculated. The results show a significant improvement for three algorithms on star-shaped graphs. Ang-GA has a higher running time as it goes through all the adjacent edges (those edges that share a node).

Ang-GA does not have good results for random graphs and this is because the defined fitness function for Ang-GA is more suitable for star-shaped graphs or graphs with some long branches such as trees. However the Dist-GA and SA algorithms report better results. Tables 5.10 and 5.11 show the average details of all of the experiments (\(p\)-value and running time).

Table 5.10 Statistical results of \(p\)-value with three different algorithms on three different data-sets.

<table>
<thead>
<tr>
<th>(p - Value)</th>
<th>(Dist - GA)</th>
<th>(Ang - GA)</th>
<th>(SA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star - Shaped</td>
<td>0.000178</td>
<td>3.8828E - 05</td>
<td>0.00026</td>
</tr>
<tr>
<td>Cactus</td>
<td>0.684E - 16</td>
<td>9.9764E - 08</td>
<td>0.00047</td>
</tr>
<tr>
<td>Random - Graph</td>
<td>0.0228</td>
<td>0.08461</td>
<td>0.03532</td>
</tr>
</tbody>
</table>
Table 5.11 Statistical results of running time (Sec) with three different algorithms on three different data-sets.

<table>
<thead>
<tr>
<th>RunningTime</th>
<th>Dist – GA</th>
<th>Ang – GA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star – Shaped</td>
<td>0.521</td>
<td>1.203</td>
<td>0.105</td>
</tr>
<tr>
<td>Cactus</td>
<td>0.521</td>
<td>1.066</td>
<td>0.104</td>
</tr>
<tr>
<td>Random – Graph</td>
<td>0.385</td>
<td>2.287</td>
<td>0.073</td>
</tr>
</tbody>
</table>

5.9 Conclusion

In this chapter a novel approach to creating a smart initial node placement for incorporation with any single-level iterative force directed graph drawing algorithm is presented. Our approach optimizes the quality of the graph layout, however it is not involved in the actual drawing process directly but instead helps the actual graph drawing algorithm to start with a better state. While the quality of a layout depends on several different aesthetic criteria what we measure in this work, as the quality of a layout, is just the number of edge crossings. Our new approach proposes two genetic algorithms and one simulated annealing algorithm. The two genetic algorithms are typically more expensive in terms of running time, while simulated annealing, in contrast, has a better running time with results not as good as the genetic algorithms but still reasonable. Our proposed solution, as mentioned earlier, is more suitable for small graphs. This is because evolutionary algorithms are expensive in terms of running time for large graphs. In this work we have shown that graphs normally suffer from a bad INP. Our results have shown a significant improvement when one of our proposed algorithms is used. Our aim here is not to find the best evolutionary algorithm as a solution to optimize the problem of edge crossings but to show that evolutionary algorithms with the aforementioned fitness functions can result in significant improvements in this area. The running time of all aforementioned algorithms (GAs and SA) depend on the size of the graph. This means the running times of the proposed optimization algorithms are high when the graph is large. Therefore, while also applicable to larger graphs, we believe that for smaller graphs, which have short running time, the use of our approach can be greatly beneficial in terms of overall reduction of edge crossings (see Section 5.8).
Chapter 6

Conclusion

6.1 Overall View

In this thesis we look at the field of graph drawing from a novel point of view. Generally speaking, a graph drawing algorithm is assessed by both the quality of the layouts it generates and the time necessary for computing these layouts [94, 95, 95, 54, 22, 36]. Improvement in one parameter is usually at the cost of the other [39]. Our main focus in this thesis is to pay sufficient attention to both parameters in order to create a balance between them. Optimizing the energy or stress of a layout is usually a time consuming process in most of the state-of-the-art graph drawing algorithms. We introduce a new concentric approach to the field of graph drawing along with some different algorithms that can draw any connected undirected graph without needing to take the stress or the energy of the layout into account. The concentric approach can optimize the running time while the quality of the layout is comparable with the quality of the state-of-the-art algorithms.

The concentric approach falls into two main categories based on the type of the input data set:

1. Drawing graphs using a distance matrix (algorithms CTD and CMD).
2. Drawing graphs using an adjacency matrix (algorithms sync and burst, CBD).

There are multiple algorithms for drawing graphs using its distance matrix [22, 67, 70, 39]. We compare the CTD algorithm to Stress Majorization [39] which is the algorithm most similar to it. Our experiments report running times 15 times faster than the Stress Majorization on average. The idea of the concentric approach encouraged us to bridge into the field of multidimensional data visualization when a dissimilarity
matrix of the data is available. We call the variation of our algorithm which deals with multidimensional data CMD, and compare CMD to the two well-known algorithms in this field, t-SNE and the classical MDS.

The concentric approach is not limited to utilizing the distance matrix of a graph, it can equally well also work with the adjacency matrix of a graph. The model we propose in Chapter 4 is a combination of two algorithms (sync and burst) which work with the adjacency matrix of a graph within the concentric approach.

One of the most distinguished features of the concentric approach is omitting the phase of calculating physical distances utilized for minimizing either the energy or the stress of a layout as typically done by the state-of-the-art graph drawing algorithms. We believe that the concentric approach brings a fresh perspective to the field of graph drawing by showing how the running time of the graph drawing algorithm can be minimized without compromising the quality of layouts, and even achieving better quality in some cases as demonstrated in this thesis. The original idea of this work comes from our publication [119] that simulates a dynamics process on networks. That work led us to the investigation of how to draw a graph without the use of stress or energy.

We also address the problem of local-minima entrapment of graph drawing algorithms which lay out graphs by minimizing a certain parameter of the layout. We propose a solution by introducing a pre-processing step in the form of an evolutionary algorithm which also helps avoiding local-minima. We show that such a pre-processing step is also beneficial within the proposed concentric approach. Because of its time complexity, this pre-processing step is particularly useful when drawing relatively small graphs.

### 6.2 Relationship to the other Research Work

The proposed algorithms in this thesis are comparable with some other works in the field of graph drawing. The similar class of graph drawing algorithms to our proposed algorithms is the class of force-directed algorithms. Algorithms in this class are iterative algorithms. They start from an initial node placement and refine the placement over a number of iterations. Almost all of the algorithms that belong to this class make use of energy or stress in order to optimize the layout. In the following sections the similar works to our work are discussed separately.
6.2 Relationship to the other Research Work

6.2.1 Work related to the CTD Algorithm

The CTD algorithm is a novel graph drawing algorithm based on the concentric approach that makes use of distance matrix of a given graph in order to visualize any undirected graph. The distance matrix is calculated using the adjacency matrix of a graph. The pioneering graph drawing algorithm to visualize graphs with their distance matrix is the one by Kamada and Kawai [70] (KK). KK is an algorithm that works based on stress minimization. Stress in this algorithm represents the differences between the theoretical and Euclidean distances on each pair of nodes. Stress Majorization (SM) is another algorithm based on theoretical distances. SM has advantages over the KK algorithm in terms of running time and stability [39]. The CTD algorithm is comparable with the SM model for two reasons. Firstly both algorithms make use of the distance matrix of a given graph. Secondly both algorithms consider the interaction (force or stress calculation) between all pairs of nodes not only those pairs with distances shorter than a predefined threshold, such as the algorithm by Harel and Koren [54]. SM makes use of conjugate gradient in order to solve their equation that optimizes the stress of the layout. The number of required iterations in SM is varied. The algorithm terminates when \( \frac{\text{stress}(X(t)) - \text{stress}(X(t+1))}{\text{stress}(X(t))} < \epsilon \) where \( \epsilon \approx 10^{-4} \) and \( \text{stress}(X(t+1)) \) represents the stress of the layout at the time \( t + 1 \). The CTD algorithm terminates after a predefined number iterations that is related to the size of a given graph. Our results in Chapter 3 illustrate that CTD outperforms SM in running time and layout quality. SM starts with a RINP that might cause local-minima. The CTD algorithm in contrast, makes use of the multilevel paradigm in order to decrease the probability of local-minima. For instance the layout in Figure 3.17b shows two layouts for the \textit{nasa2146} graph by both SM and CTD. SM might end up with different layouts every time as it starts with a RINP every time while CTD starts with specific nodes at the beginning.

6.2.2 Work related to the CMD Algorithm

The CMD algorithm is similar to the CTD algorithm in a sense that both algorithms make use of a square matrix that represents the distance between all pairs of nodes (CTD) or individual data items (CMD). CTD is a graph drawing algorithm therefore it uses the distance matrix (theoretical distance matrix) of a given graph while CMD is a multidimensional data visualization algorithm that uses the dissimilarity matrix that represents dissimilarities between each pair of data individuals. There are some other algorithms that were developed in order to deal with the multidimensional
data visualization such as the class of *Multidimensional Scaling* (MDS), Sammon’s projection [105] and t-Distributed Stochastic Neighbor Embedding (t-SNE) by Maaten *et al.* [85]. t-SNE is a recent developed algorithm and it is the state-of-the-art in the field of multidimensional visualization. We compare the CMD algorithm with t-SNE and one of the algorithms belonging to the MDS class (Classical MDS).

t-SNE makes use of the similarity between a given pair of nodes \((n_i \text{ and } n_j)\) in order to create a conditional probability. Then \(n_i\) picks \(n_j\) as its neighbor based on this probability. t-SNE can produce different visual results for the same data set after each run. This happens due to the conversion of similarity to probability.

The CMD algorithm is comparable with t-SNE in both running time and the quality of the visual results. The CMD algorithm takes around 20 seconds to visualize a data set with around 2000 individuals while t-SNE needs around 4 minutes for the same data set. Maaten *et al.* suggests to measure a visual layout by using the available information about the data set. To do so we decided to create a few artificial data sets and visualize them using three different approaches: CMD, t-SNE and the classical MDS. The strength of our proposed algorithm over the state-of-the-art (t-SNE) and the classical MDS is shown in Figures 3.34 and 3.35. The CMD algorithm is a fast algorithm in terms of running time compare with t-SNE.

### 6.2.3 Work related to the CBD Model

The CBD model is a graph drawing model that contains two separate algorithms, sync and burst. This model draws any undirected and connected graph by making use of the graph’s adjacency matrix without calculating the distance matrix. The closest algorithms to our model is the one by Hu [60] and the binary stress model (BStress) by Koren *et al.* [82]. These two models draw any undirected and connected graph by making use of the adjacency matrix.

**Comparison to the Hu Model**

The Hu model is one of the best known and fastest graph drawing algorithms. This is a multilevel algorithm and it is implemented in Graphviz [41]. The two factors in graph drawing that the multilevel paradigm helps to improve on are the local-minima and the running time [9, 122, 60]. Unlike the CBD model, the Hu model draws undirected graphs by minimizing the energy of the system - that is the main difference between the Hu model and the CBD model. Another difference between the Hu model and the CBD model is that the CBD model is not a multilevel algorithm. Since there is no
force or theoretical distance involved in our proposed model CBD can make use of the idea of the quad tree technique while the Hu model is not able to do that. Unlike the Hu model, the CBD model uses all the nodes starting from the first iteration without applying any coarsening techniques on the nodes. However, although the Hu model and CBD have almost similar running times, the quality of their results are different. The CBD model outperforms the Hu model in terms of the layout quality. For instance the layout of the Oberwolfach-filter2D graph in Figure 4.15a is not able to illustrate the details of the graph. This graph has 6 large holes around the graph. The Hu model is only illustrates 4 of them. The layouts by CBD in Figure A.27 (the one by only sync and the one by sync and burst) are able to show those details about the holes. The other example is the finance256 graph by the Hu model in Figure 4.16a. This graph has 8 rings connected to a circle. All those 8 rings have identical structure while the layout by Hu in Figure 4.16a does not imply this. The layouts by the CBD model in Figure 4.35 is able to show 8 identical rings especially in the layout by only the sync algorithm. The layout for the skirt graph by the Hu model in Figure 4.18a is another example that the details of the graph is not clear enough. There are some connections inside of each internal area that are not visualized properly by the Hu model while those connections are visible clearly by the CBD model in Figure 4.39.

**Comparison to the BStress Model**

The BStress model by Koren *et al.* [82] is the other graph drawing algorithm that makes use of the adjacency matrix without calculating the distance matrix in order to draw any undirected graph. This model is similar to our model mostly because both models are able to visualize graphs within a circular shape. The CBD model, as mentioned earlier, has two algorithms (sync and burst). The sync algorithm is the first algorithm and the burst algorithm has to come after the sync algorithm. The burst algorithm is responsible for drawing the graph in a circular shape. The BStress model makes use of a quad tree to speed up the calculations. The total required number of iterations in the BStress model is lower than the total number of iterations in the CBD model. The type of calculations in BStress model is different in contrast to the type of calculations in the CBD model. The BStress model has two expensive (in terms of running time) steps at each iteration. The conjugate gradient that solves the two equations $A \times x = bX$ and $A \times y = bY$ is used in the BStress model at each iteration where $bX$ and $bY$ are two vectors that represent the accumulated *sine* and *cosine* values associated with each node, and $A$ is a square matrix that is created by
the adjacency matrix. The approximation technique by Barnes et al. [7] (QT) is also used in BStress in order to calculate \( bX \) and \( bY \). These two steps (conjugate gradient and QT) are two expensive calculations in terms of running time and they need to be performed at each iteration. In contrast with the CBD model, each single iteration in the BStress model involves considerably more calculation. First of all the CBD model has no linear equation to solve similar to those in the BStress model; e.g. \( A \times x = bX \) and \( A \times y = bY \). The common calculation step between our model and BStress model is to approximate the accumulated sine and cosine. The BStress model makes use of the QT technique by Barnes et al. [7] and the CBD model makes use of the idea of the QT technique together with a new technique in order to approximate sine and cosine values. Although this step (approximation of accumulated sine and cosine) is the only common calculation between the CBD and BStress model, the CBD model does not need to perform this calculation at each iteration. For example for the plastk10 graph with around 80k nodes there are roughly 20 iterations needed to perform approximation of accumulated sine and cosine while the total number of iterations in BStress model that needs to approximate the accumulated sine and cosine for the same graph is 159 [82]. The other calculation that is needed to be performed at each iteration in the CBD model is a very fast calculation. This calculation, for each node \( n_i \), computes the average of the positions of all nodes adjacent to \( n_i \).

However the BStress and CBD models both tend to draw a given graph within a circular shape but BStress model is not always successful in terms of showing the structure or the details of a given graph. One of the features of the BStress model is that it tends to fill the interior of the circular drawing area. This feature sometimes is not desirable as the layout does not exactly represent the structure of the graph. Figure 6.1 shows two layouts by BStress and CBD of a graph with 1000 nodes and 1000 edges. As you can see the layout on the left in Figure 6.1 by BStress insists in filling the circular area while the layout on the right by CBD illustrates the structure of the graph which is simply a circular graph (closed circle).
6.3 Future Work

The concentric approach is a new way of drawing connected undirected graphs and the results presented in this thesis prove its usability. This approach has the characteristics of an algorithmic framework which can accommodate a variety of techniques, such as the multilevel paradigm adoption, smart initial layouts, etc.. We believe there is still room to research on different aspects of this approach and optimize it further. Moreover, we believe that this approach, potentially, is extensible for drawing directed graphs. An edge \( e \in E \) in directed graph is an ordered pair of nodes \((n_i, n_j)\) that \(n_i\) is the source and \(n_j\) is the head of the edge. The proposed concentric approach opposes each pair of nodes (undirected) on the border of a single circle in front of each other. To draw a directed graph one can oppose a pair of nodes on the border of two circles centered at the origin of the Cartesian system. Figure 6.2 shows two examples of using the concentric approach. Figure 6.2a is the concentric approach that is already used and proposed in this thesis and suitable for undirected graphs. Figure 6.2b shows a potential concentric approach in order to be used for directed graphs when the source and head of all edges are known.
The last chapter of this thesis (see Chapter 5) explores the use of the evolutionary algorithms on small graphs as a pre-processing step in order to reduce the chance of edge crossings. Our aim in that chapter is to show that the evolutionary algorithms can help to improve the quality of the layouts for small graphs. More research can be carried out in this field in order to improve our current results in terms of layout quality and running time. The evolutionary algorithms that have been presented in this thesis work well with small size graphs for the reason of running time efficiency. However an evolutionary algorithm using simpler techniques and with a less complicated fitness function can be modeled and used for large graphs as well.

Most of the presented algorithms have been implemented in Java and some of the jar files are available at http://graphvision.org/. We are currently working on a visualization framework to include all the presented algorithms in this thesis into one unique visualization framework.
References


[87] Martz, E. (2015-12-03). What can you say when your p-value is greater than 0.05? http://blog.minitab.com/blog/understanding-statistics/what-can-you-say-when-your-p-value-is-greater-than-005.


Appendix A

A.1 A list of Figures for Chapter 3 by CTD, SM and TASC

A.1.1 CTD vs SM

Each graph is visualized both by the CTD algorithm (presented in Chapter 3) and the Stress Majorization (SM).

Fig. A.1 shyy-41 graph from [61] with their details in Table 3.4.
Fig. A.2 3 different graphs from [61] with their details in Table 3.4.
A.1 A list of Figures for Chapter 3 by CTD, SM and TASC

Fig. A.3 3 different graphs from [61] with their details in Table 3.4.
Fig. A.4 3 different graphs from [61] with their details in Table 3.4.
Fig. A.5 3 different graphs from [61] with their details in Table 3.4.
Fig. A.6 2 different graphs from [61] with their details in Table 3.4.
A.1.2 CTD using TASC

Each graph is visualized two times by the CTD (with and without the TASC).

(a) monien-grid2 by CTD.

(b) monien-grid2 by CTD using TASC.

(c) Sierpinski with 3282 nodes by CTD.

(d) Sierpinski with 3282 nodes by CTD using TASC.

Fig. A.7 2 different graphs from [61] with their details in Table 3.5 using Pythagoras and TASC.
Fig. A.8 2 different graphs from [61] with their details in Table 3.5 using Pythagoras and TASC.
A.1.3 CTD on Small Graphs using SINP

Each graph is visualized both by the CTD algorithm (presented in Chapter 3) as a sequence of layouts (starting from the smart initial layout) and the Stress Majorization (SM).

(a) A star-shaped graph with 6 branches by CTD using SINP with 33 nodes and 69 edges.

(b) A star-shaped graph with 6 branches by SM with 33 nodes and 69 edges.

Fig. A.9 Star-shaped graph.

(a) A Tree graph by CTD with 50 nodes and 49 edges.

(b) A Tree graph by SM with 50 nodes and 49 edges.

Fig. A.10 Tree graph.
(a) A Tree by CTD with 88 nodes and 87 edges.
(b) A Tree by SM with 88 nodes and 87 edges.

Fig. A.11 Tree graph.

(a) A stare-shaped graph by CTD with 16 nodes and 33 edges.
(b) A stare-shaped graph by SM with 16 nodes and 33 edges.

Fig. A.12 Star-shaped graph.
A.2 A list of Figures for Chapter 4 by CBD model applying Sync and Burst Algorithms

A.2.1 CMD vs BStress

Each graph is visualized by the sync and the burst and the BStress model (presented in Chapter 4).

Fig. A.13 Star-Shaped graph with 39 nodes and 45 edges. (see Table 4.3 number 1).

Fig. A.14 Star-Shaped graph with 33 nodes and 69 edges. (see Table 4.3 number 4).
Fig. A.15 The Heawood graph with 14 nodes and 21 edges. (see Table 4.3 number 5).

Fig. A.16 The Peterson graph with 10 nodes and 15 edges. (see Table 4.3 number 7).

Fig. A.17 Queen graph (8 X 8) with 64 nodes and 728 edges. (see Table 4.3 number 8).
A.2 A list of Figures for Chapter 4 by CBD model applying Sync and Burst Algorithms

Fig. A.18 Tree with 90 nodes and 89 edges. (see Table 4.3 number 3).

(a) A layout by the sync algorithm.  
(b) A layout by the sync and burst algorithms.  
(c) A layout by the BStress.

Fig. A.19 Mesh 20X100, nodes: 2000, edges: 3880. (see Table 4.4 number 3).

(a) A layout by the sync algorithm.  
(b) A layout by the sync and burst algorithms with 5 iterations.  
(c) A layout by the sync and burst algorithms with 15 iterations.
Fig. A.20 The \textit{ag-monien} graph, nodes: 2559, edges: 4092. (see Table 4.4 number 5).

Fig. A.21 The graph \textit{ag-monien-grid2}, nodes: 3136, edges: 6112. (see Table 4.4 number 6).
Fig. A.22 The graph *ag-monien-stufe*, nodes: 1036, edges: 1868. (see Table 4.4 number 7).

Fig. A.23 The *utm3060* graph, nodes: 3060, edges: 28880. (see Table 4.4 number 11).
(a) A layout by the sync algorithm.

(b) A layout by the sync and burst algorithms with 5 iterations.

(c) A layout by the sync and burst algorithms with 15 iterations.

Fig. A.24 The wang-swang2 graph, nodes: 3169, edges: 8836. (see Table 4.4 number 12).

(a) A layout by the sync algorithm.

(b) A layout by the sync and burst algorithms with 5 iterations.

(c) A layout by the sync and burst algorithms with 15 iterations.

Fig. A.25 The Sierpinski graph with 1095 nodes and 2187 edges.(see Table 4.4 number 13).
Fig. A.26 A layout for the gearbox graph by the sync algorithm with 153746 nodes and 4463329 edges. (see Table 4.4 number 21).
Fig. A.27 The Oberwolfach-filter2D graph with 1668 nodes and 4541 edges. (see Table 4.4 number 23).

(a) A layout by the sync algorithm.
(b) A layouts by the sync and burst algorithms with 3 iterations.

Fig. A.28 The lung2 graph with 109460 nodes and 273646 edges. (see Table 4.4 number 25).

(a) A layout by the sync algorithm.
(b) A layouts by the sync and burst algorithms with 10 iterations.
Fig. A.29 The AG-Monien-3elt graph with 4721 nodes and 13722 edges.
A.2.2 3D visualization by CMD and BStress

A few examples of 3D visualizations by the CBD and BStress algorithms.

(a) Two different views of a 3D visualization for a graph with 39 nodes by CBD using a sphere to show how close nodes are to the surface of the sphere.

(b) Two different views of a 3D visualization for a graph with 39 nodes by the CBD algorithm without using sphere.

(c) Two different views of a 3D visualization for a graph with 39 nodes by the BStress algorithm using a sphere to show how close nodes are to the surface of the sphere.

Fig. A.30 Star-Shaped graph with 39 nodes and 45 edges. (See Table 4.3 number 1).
A.2 A list of Figures for Chapter 4 by CBD model applying Sync and Burst Algorithms.

(a) Two different views of a 3D visualization for Sierpinski graph with 123 nodes by the CBD algorithm using a sphere to show how close nodes are to the surface of the sphere.

(b) Two different views of a 3D visualization for Sierpinski graph with 123 nodes by the CBD algorithm without using sphere.

(c) Two different views of a 3D visualization for Sierpinski graph with 123 nodes by the BStress algorithm using a sphere to show how close nodes are to the surface of the sphere.

Fig. A.31 *Sierpinski* graph with 123 nodes and 243 edges. (See Table 4.3 number 2).
(a) Two different views of a 3D visualization for a graph (tree) with 90 nodes by the CBD algorithm using a sphere to show how close nodes are to the surface of the sphere.

(b) Two different views of a 3D visualization for a graph (tree) with 90 nodes by the CBD algorithm without using sphere.

(c) Two different views of a 3D visualization for a graph (tree) with 90 nodes by the BStress algorithm using a sphere to show how close nodes are to the surface of the sphere.

Fig. A.32 Two different views of a 3D layout for a tree with 90 and 89 edges by the CBD algorithm and the BStress.
Index

Authors
Ali Civil, 27
Andre M.S Barreto, 34
David Harel, 26
Edward M Reingold, 23
Emden R. Gansner, 26
Jürgen Branke, 34
Laurens van der Maaten, 43
Peter Eades, 23
Ron Davidson, 36
Satoru Kawai, 25
Thomas M. J Fruchterman, 23
Tomihisa Kamada, 25
Yehoda Koren, 27
Yifan Hu, 22

Big data, 1
Concentric Approach, 46
CMD, 88
CTD, 54
Cosine Similarity, 94
Emden Ganser, 40

Genetic Algorithm, 171
Crossover, 173
Elitism, 173
Fitness Cost, 172
Fitness Function, 172
Mutation, 173

Reproduction, 173
Selection, 172
Tournament Selection, 172
Giacomo, 39

Graph, 3
Adjacency Matrix, 16
Adjacent Nodes, 15
Betweenness Centrality of Node, 17
Connected Graph, 16
Degree of Node, 16
Directed Graph, 15
Distance Matrix, 17
Edge List, 16
Incident Edge, 15
node-count, 186
Path in Graph, 16
Queen Graph, 141
Shortest Path, 16
Undirected Graph, 15
Weighted Graph, 16

Graph Layout, 18
Attraction Force, 19
Cartesian Coordinates, 18
Edge Crossing, 18
Euclidean Distance, 18
INP, 22
Repulsion Force, 19

Iterative Method, 19
Multidimensional Data, 5
  Dissimilarity Matrix, 17

N-body, 20

p-value, 185
Pairwise Information, 18
Power Law, 19

Riemannian, 41

Simulated Annealing, 182
Software
  GraphViz, 160
  sfdp, 160
Sugiyama, 39

Techniques
  Centering Matrix, 21
  Cholesky factorization, 27
  Floyd-Warshall, 26
  Johnson’s algorithm, 26
  Multilevel Paradigm, 20
    Node Coarsening, 20
  Power Iteration, 21
  Quad Tree, 21
  Simulated Annealing, 182
  Trigonometrical Approximation, 22

Visualization methods
  Dendrogram, 2
    Aesthetic Criteria, 19
    BStress, 27
    FR, 23
  Hasse Diagram, 39
  Hu, 31
  KK, 25
  Layered Graph Drawing, 39

Local-Minima, 20
  Maxent, 40
  Radial Graph Drawing, 39
  Ring Diagram, 39
  SDE, 38
  SM: Stress Majorization, 26
  Sparse Stress Model, 40
  SSDE, 38
  TimoGA, 35
  Hasse Diagram, 21
  heat map, 2
  MDS, 38
  Sammon Projection, 42
  scatter, 2
  t-SNE, 43
  VSM, 94

Well-known Graphs
  ag-monien-ukerbe1, 151
  finance256, 151
  gset-g34, 151
  heawood, 141
  jagmesh8, 151
  peterson, 141
  sierpinski, 151
  wagner, 141
  wang-swang2, 151