

## **ABSTRACT**

Graph Visualization is an interdisciplinary technology which aims to visualize data information and relationship, it's an effective and intuitional way to study abstract data. There are a wide range of related subjects: graph Theory, graph drawing algorithm, human-computer interaction, computer science, aesthetics and so on. Along the development of these subjects, especially the rapid development of computer science, accelerate graph visualization's growth.

In this thesis, we study on the most popular graph representation: node-link layout. We first introduce graph, drawing graph according to input data, relationship between graph components and data (graph theory). Then we study on graph layout features and algorithms, focus on revealing graph components relationship from data operation algorithms. Specially, we present a new method for graph drawing: partially drawn links. In the end, briefly introduce our interactive interface and functions' performance test, discussion of what we did and future work.

<b>Table of Contents .....</b>	<b>I</b>
<b>Table of Figures.....</b>	<b>III</b>
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Origin.....	1
1.2 Node-Link Layout.....	2
1.3 Layout Features .....	3
1.4 Interactive Visualization .....	5
<b>Chapter 2 Graph Theory .....</b>	<b>7</b>
2.1 Graph .....	7
2.2 Adjacency Matrix .....	7
2.3 Important Algorithms in Graph Theory.....	10
2.3.1 All Pairs Shortest path .....	11
2.3.2 Depth First Search Algorithm (DFS).....	13
<b>Chapter 3 Graph Visualization/Graph Drawing .....</b>	<b>18</b>
3.1 Graph Drawing and Aesthetics .....	18
3.2 Force-directed Layout Algorithm.....	21
3.2.1 Spring embedder (Eades, Kamada and Kawai) .....	21
3.2.2 Force-directed Method (Fruchterman and Reingold 90) .....	23
3.3 Partial Drawn Links in Directed Graph (Edges Shorten) .....	27
3.3.1 Type of Edges .....	27
3.3.2 Partially Drawn Links Methods.....	30
3.3.3 Nodes Placement .....	33
3.3.4 Partially Drawn Links in User-defined Region.....	36
3.3.5 Discussion and Further work .....	37
3.4 Hierarchical Clustering Algorithm .....	38
3.4.1 Agglomerative Clustering .....	38

3.4.2 Linkage Criteria.....	39
3.4.3 Directed Graph.....	42
3.5 Strongly Connected Components of Directed Graph.....	43
3.5.1 Strongly Connected Components .....	43
3.5.2 Searching Algorithm.....	43
3.5.3 Graph Layout of SCC .....	45
3.6 Shortest Path between Two Nodes.....	47
<b>Chapter 4 Matlab GUI Instructions .....</b>	<b>49</b>
<b>Chapter 5 Conclusion, Discussion and Future Work .</b>	<b>52</b>
5.1 Performance .....	52
5.2 Discussion and further work.....	52
<b>Erklärung .....</b>	<b>54</b>
<b>Acknowledgements .....</b>	<b>55</b>
<b>Bibliography .....</b>	<b>56</b>

## Table of Figures

Figure 1.1: a) Euler`s drawing of Konigsberg Problem .....	1
b) reduced diagram.....	1
Figure 1.2: Social Network (from FaceTru.com) .....	2
Figure 1.3: A directed graph with 30 nodes and 106 edges.....	3
Figure 1.4: Force-directed layout with 30 nodes and 106 edges .....	4
Figure 1.5: partially drawn links .....	4
Figure 1.6: Hierarchical clustering .....	5
Figure 1.7: Graph visualization reference model <sup>[25]</sup> .....	6
Figure 2.1 a): Adjacency Matrix(directed) .....	8
b): Directed Graph .....	8
Figure 2.2 a): Adjacency Matrix(undirected) .....	9
b): Undirected Graph .....	9
Figure 2.3 a): adjacency matrix with self-connection .....	10
b): 's' in the node id to represent self-connection.....	10
Figure 2.4: Intermedia nodes.....	12
Figure 2.5: Example of Floyd-Warshall.....	13
Figure 3.1 Subway map of HongKong.....	19
Figure 3.2: S-Bahn map of Stuttgart .....	19
Figure 3.3: a) minimizing edge crossing layout .....	21
b) maximizing symmetry layout .....	21
Figure 3.4: replace edge with spring.....	21
Figure 3.5: force between nodes .....	21
Figure 3.6: Spring Algorithm (Eades).....	23
Figure 3.7: forces versus distance .....	24
Figure 3.8: Force-directed placement.....	25
Figure 3.9: Grid boxes technique.....	26

Figure 3.10: modeling the frame along x-coordinate .....	26
Figure 3.11: Graph of email record .....	27
Figure 3.12: a) all straight edges .....	28
b) curved edge instead.....	28
Figure 3.13: a) all straight edges; .....	28
b) with curved edge.....	28
Figure 3.14: Bezier curve .....	28
Figure 3.15: Orthogonal grid drawing .....	29
Figure 3.16: polyline upward grid drawing .....	29
Figure 3.17: a) street sign at Dublin.....	30
b) graph representation.....	30
Figure 3.18: a) x-y coordinate of link .....	31
b) intermedia node introduced .....	31
Figure 3.19: location relationship between start node and end node....	32
Figure 3.20: x,y absolute value comparison .....	32
Figure 3.21: Length control.....	33
Figure 3.22: failure example .....	33
Figure 3.23: a) Force-directed layout.....	34
b) partially drawn links .....	34
Figure 3.24: circular placement .....	35
Figure 3.25: a) before shorten links .....	35
b) after links shortened .....	35
Figure 3.26: a) region selected .....	37
b) partially drawn links for region .....	37
Figure 3.27: Flow chat of hierarchical clustering.....	39
Figure 3.28: Single linkage .....	40
Figure 3.29: Complete linkage.....	40

Figure 3.30: Average linkage .....	40
Figure 3.31: Undirected graph .....	43
Figure 3.32: Graph size reduce .....	46
Figure 3.33: Color coding layout.....	46
Figure 3.34: Shortest path between two nodes .....	47
Figure 3.35: Shortest Path from City C to City F.....	48

# Chapter 1 Introduction

## 1.1 Origin

Graph is an intuitional model to visualize interconnection. it can provides information like: communication degree, relation degree, communicate direction(who contact who) and frequency. We can use human eyes to see, explore abstract information without mathematical analysis. It's a convenient way to describe some instances, similarity (dissimilarity), depth and relationship.

In 1736, Euler published originating graph theory. In his theory which used to solve the famous Konigsberg problem(7 bridges), Euler believed that the key point is the number of bridges which connect banks and islands, has nothing to do with size or shape of banks and islands, or length of bridges. He regarded banks and islands as nodes to be visited, and bridges as links which connect corresponding vertex. Therefore, a graph was been drawn, Konigsberg problem is transform to the question: whether you could 'drawing this picture without retracing any line and without picking your pencil up off the paper'<sup>[2]</sup>.

Despite the first graph was published in early 1736, the development is actually in recent decades. Knuth`s paper about computer-drawn flowcharts<sup>[3]</sup> is generally acknowledged as the first time introduce layout algorithm into visualization science.

The generation of graph has important applications in many fields, like social network, internet communications, map routes, enables users to see and understand large amounts of information. Because of that, graph layout must has readability. There`re different graph representations: node-link layout, space nested layout, 3D layout and so on<sup>[1]</sup>.

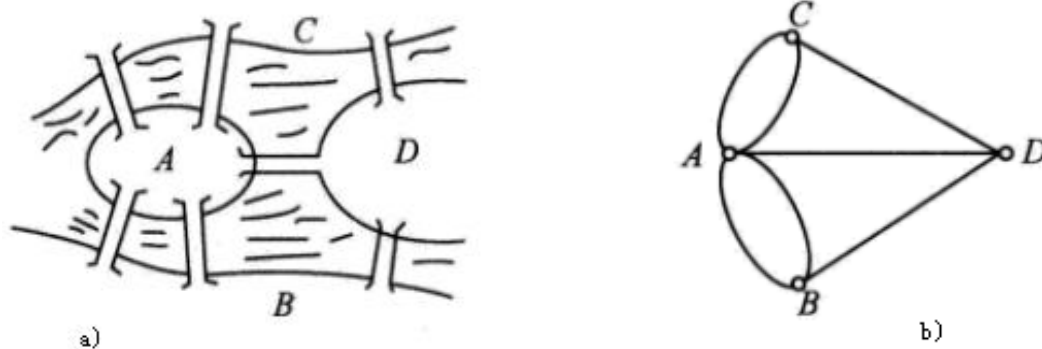


Figure 1.1: a) Euler`s drawing of Konigsberg Problem

b) reduced diagram

## 1.2 Node-Link Layout

Node-link diagram is the most popular layout, it's nothing but a collection of nodes links in which nodes represent entities and link between two nodes means they have correlation, more internodes information can be observed along with different techniques. It arranges nodes and links in a space to help us to understand the whole and regional data information.

'Many node-link graphs/diagrams use boxes or circles to show entities and linking lines drawn between them to represent relationships between the entities.'<sup>[6]</sup> For example, in internet social network(like Facebook), each person can be represented as a node, connections between them are links(edges). In figure 1.2, labels on links describe their relationship and red labels around the nodes represent activities occurred between pairwise nodes. Interpersonal relationship, communication and activities can be observed easily, assign different components with different colors enhance heuristics of layout. Besides, use photos as node ids make the relationship between pairwise nodes more clear, won't confuse about which one is 'Father' for instance.



Figure 1.2: Social Network (from FaceTru.com)

There're two kinds of graphs: directed and undirected graph. Obviously, 'directed' means connection has direction, visualizator need to concern more when doing data analysis and layout a graph, it's more complex than undirected graph but exhibits more information as well. Of course, which graph we use depend on the task.



### 1.3 Layout Features

Although node-link layout is the most popular model and it match human perception, when the number of nodes and links become larger, it's difficult to locate or find the links between two nodes. Figure 1.3 illustrates a directed graph with just 30 nodes and 106 edges, it's enough to confuse user to separate pairwise nodes and edge between them, difficult to identify even the arrow head(direction). This drawback forces us to develop different layout features and methods to help user to discover significant information.

In the past decades, various algorithms have been proposed, for example, force-directed placement which presented by Fruchterman<sup>[4]</sup>. The resulted spring layout is in an aesthetically pleasing way(Figure 1.4). Edges are all straight lines, uniform nodes distribution and symmetry.

We barely get information from figure 1.3 deal to overlapped nodes and inter-crossed links. It's almost impossible to find pairwise nodes by following the links. By contrast, force-directed save our eyes from chaos, user can identify node and link from mass to good looking, no more overlapped nodes, make out pairwise connected nodes by following the link become easier.

However, it requires very high running time, and both require large display space, and suffer the common problem: when data increases, user has more and more trouble to make out nodes and links.

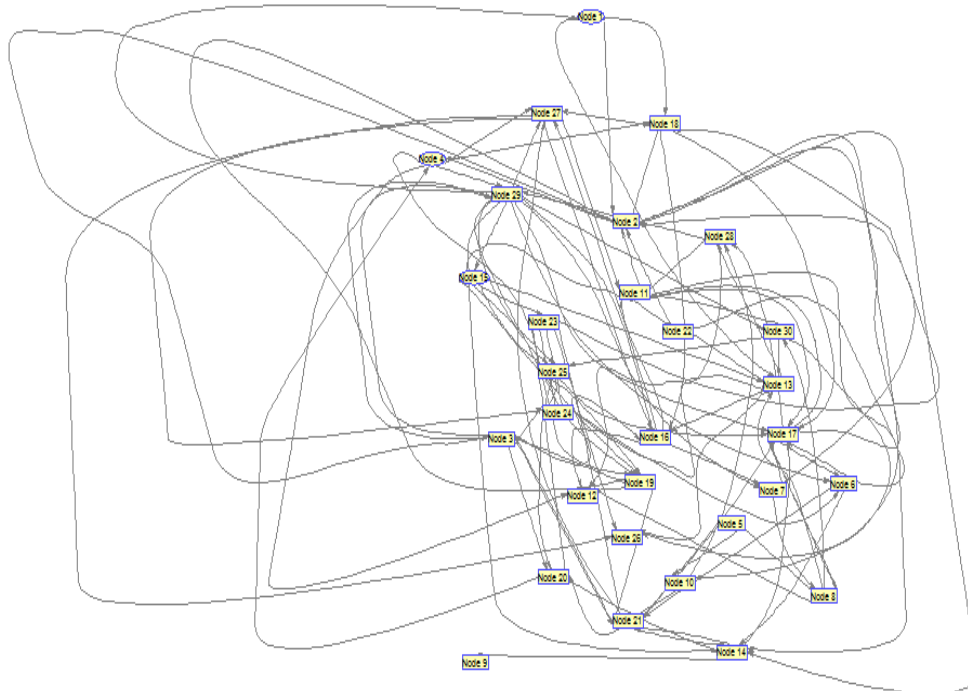


Figure 1.3: A directed graph with 30 nodes and 106 edges

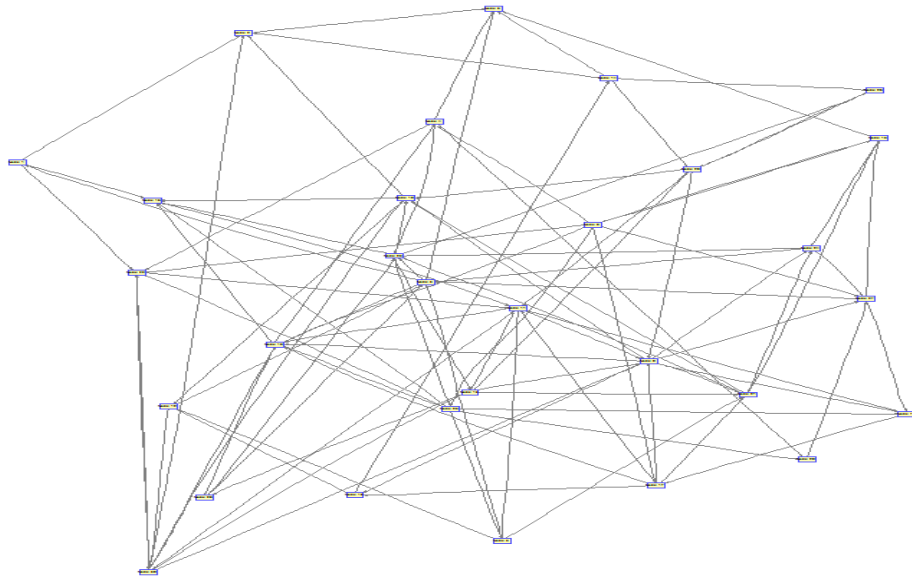


Figure 1.4: Force-directed layout with 30 nodes and 106 edges

From above comparison, we could draw a conclusion that no overlapped nodes and reduce intercrossed links make relationships between pairwise nodes more understandable. Based on this experience, we place nodes without overlap, draw links partially to reduce intersection. The arrow head on the links lead us from one node to it's connected nodes. Although the link's arrowed end doesn't connect with node closely, we can still find pairwise nodes connection under the arrow's navigation.

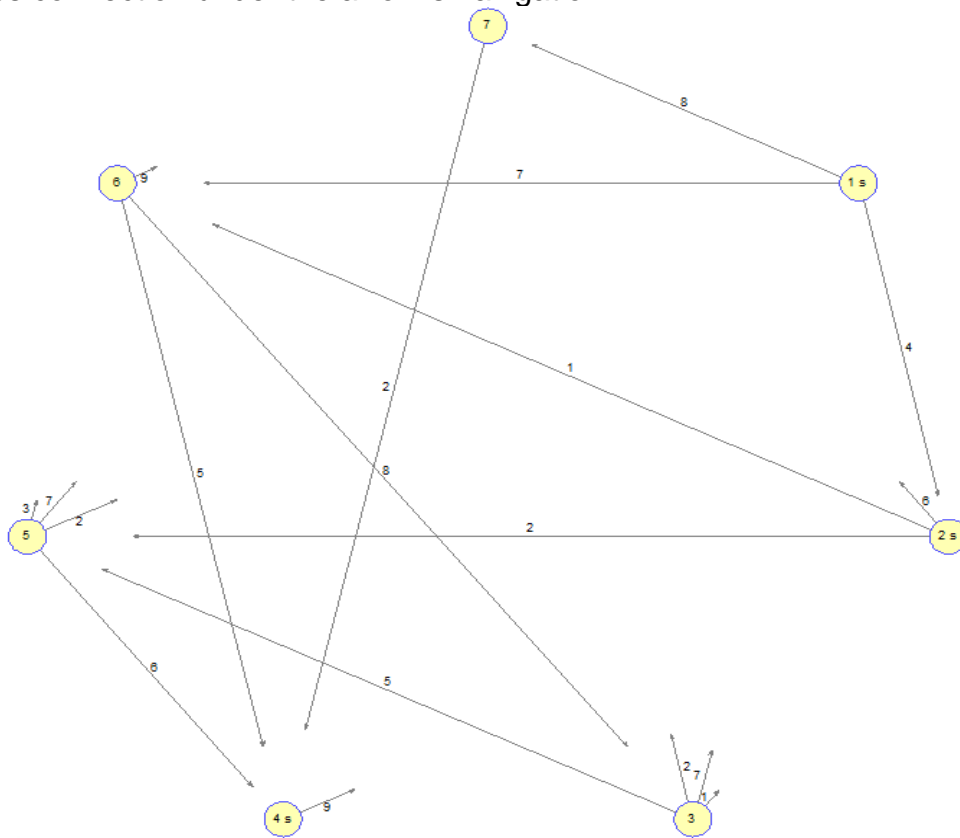


Figure 1.5: partially drawn links

Sometimes we want to sort data into subsets or clusters base on some prop-  
erties, aggregate data into different groups, or clustered in different levels.  
Members in a cluster share common property or in other word 'similarity'. For  
example, band score boundaries are set to divide examination results, a  
community can be separated to different groups base on people`s hobbies.

This community structure focus on statistical analysis of clustered groups  
which share distinctive properties regardless whether nodes in the group are  
connected by links or not. There are several types of mature clustering algo-  
rithms: exclusive clustering; overlapping clustering; hierarchical clustering,  
probabilistic clustering and so on. New algorithms are under research and  
developing also, this technique which combines graph drawing with statistics  
analysis is full of practical value. One example is social network like Facebook,  
Twitter, has demand on statistical analysis of users` interests, communication,  
regional distribution and so on, it`s not intuitive just enumerate a large amount  
of pure data.

We study in agglomerative clustering which is one version of hierarchical  
clustering algorithm. Objects are clustered with nearest distance, this cluster is  
viewed as a whole new object, distance between objects is recalculated and  
objects with new nearest distance be aggregated. After some iterations the  
final clusters formed, which is a bottom-up algorithm. The schematic diagram  
of such algorithm is illustrated in Figure 1.5.

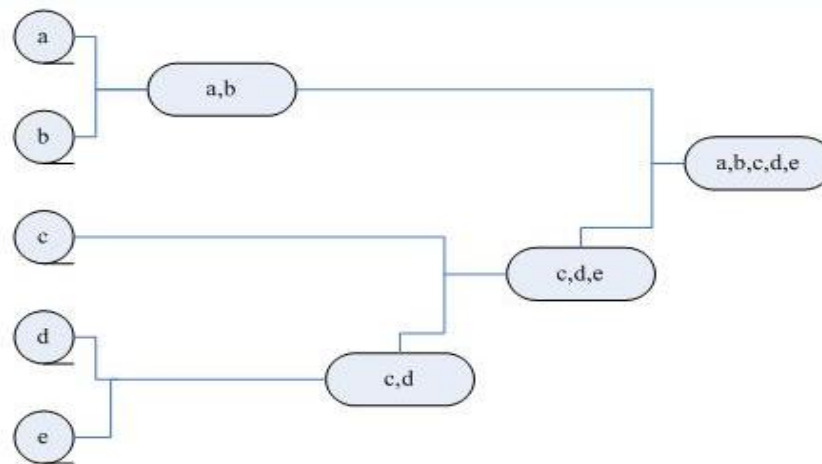


Figure 1.6: Hierarchical clustering

## 1.4 Interactive Visualization

Although we already have some layout techniques, it should be noticed that  
none of them is 'perfect', advantages coexist with disadvantages. It`s difficult  
to judge which one is 'better', more effective, more efficient, hard to generalize  
absolute criteria to evaluate results, because different algorithms deal with  
specific works with specific purposes.

By use of computer, graph visualization is actually interactive between human  
and computer. Computer is like a teacher, interpret knowledge inside the ab-  
stract data, we can ask questions to get more detailed information. The quali-  
ty(processor, graphic card, etc.) of the computer that user used also limits

visualization ability and speed. The amounts of processed data, layout axes(space), graph readability depend a lot on the computer.

We can view graph visualization as a mapping process from data to graph then to human vision perceptual system. This process is illustrated in Figure 1.6, it's a simple reference model of graph visualization. From left to right, each arrow may represent a series of transform. Arrows from 'task' to 'data'(from right to left) indicate that viewer can manipulate to adjust the transform. Graph visualization is serve the task at last.

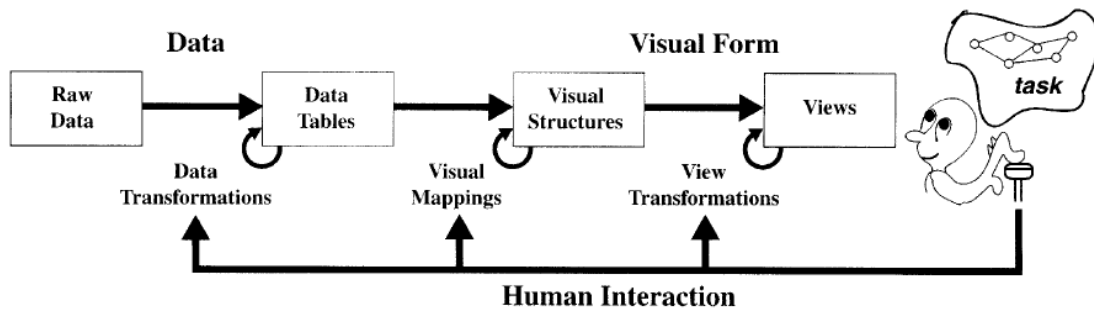


Figure 1.7: Graph visualization reference model<sup>[25]</sup>

Human-computer interactive interface is an effective way to combine different layout features, enable user to gain interested information in different aspects. For example, hierarchical clustering let us obtain data's similarity/distance/connection degree easily, force-directed or radial layout can strongly enhance the readability of trouble discerning and cluttered graph. In other words, no matter which visualization feature to be used, all aims to navigate user has better understanding from massive amount of data in different aspect at the same time.

The tasks for visualizer are clear: make data more understanding for more people, get rid of as more 'noisy' data as possible, let user choose the appropriate representations for different tasks.

There are two basic aspects of graph visualization: (1) detect, summarize, extract inside relationship and structural modeling data sets; (2) transfer structural data sets to graphical form that could be viewed and interact with. These two aspects focus on different areas but their boundary are not very clear because the same data structure can be layout in different ways<sup>[7]</sup>.

## Chapter 2 Graph Theory

### 2.1 Graph

'Graph' in the graph theory is not the graph discussed in calculus and geometry, but a mathematical abstract for the connection between some real world's entities. This graph consists of a nodes/vertices set and some links/edges, it's convenient to use graph to describe many instances in the real world. For example, regard two persons as nodes while a link connection represent they're friends; or cities as nodes and it has train between them if there is a link between them. In this kind of graph, people are more concern about whether there is a line connecting two giving nodes, not how they connect. Normally, a graph G has three components: nodes/vertices, edges/links and weight.

A graph could be undirected, means it's no distinction between nodes with connected edges. If graph is directed, edges have direction between source nodes and destination nodes.

### 2.2 Adjacency Matrix

There are several possible ways to represent a graph inside the computer, we discuss adjacency matrix in this thesis.

A graph is determined by adjacent relationship between nodes, the graph's data structure which stored in computer has to equivalent/match this relationship. An adjacency matrix is capable of this 'media' role. For a graph with N nodes can be represented by an N-by-N adjacency matrix, weights of the edges are all nonzero entries in the matrix.

For an adjacency matrix  $A = [a_{ij}]$ ,  $a_{ij} = 1$  means node i and node j has connection and, means has edge from node i to node j in directed graph. Besides,  $a_{ij} = 0$  means node i and node j are individual nodes without connection (no edge).

Further more, assign weight to each edge, an (weighted)adjacency matrix  $A = [a_{ij}]$ ,  $a_{ij}$  represents the number of edges start from node i and ended at node j in directed graph, or the number of edges which connect node i and j for an undirected graph. Besides,  $a_{ij} = 0$  means no connection(no edge) between node i and j<sup>[5]</sup>.

Adjacency matrix for a graph:

$$A[i,j] = \begin{cases} 1 & \text{if node i and node j has connection} \\ 0 & \text{if node i and node j has no connection} \end{cases}$$

(Weighted) Adjacency matrix for a graph:

$$A[i,j] = \begin{cases} w_{ij} & \text{if node } i \text{ and node } j \text{ has connection} \\ 0 & \text{if node } i \text{ and node } j \text{ has no connection} \end{cases}$$

Figure 2.1 is a weighted adjacency matrix and figure 2.2 is the corresponding directed graph. For example, we regard 'Mr. John' as node 1 and 'Mr. King' as node 2,  $a_{1,2} = 4$  could mean Mr. John sent 4 emails to Mr. King and  $a_{2,1} = 0$  means Mr. King didn't send any email to Mr. John.

$$A = \begin{bmatrix} 0 & 4 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 7 & 1 & 0 & 0 & 5 \\ 0 & 9 & 6 & 0 & 0 \\ 2 & 0 & 0 & 6 & 0 \end{bmatrix}$$

	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	0	4	0	9	0
Node 2	0	0	0	0	2
Node 3	7	1	0	0	5
Node 4	0	9	6	0	0
Node 5	2	0	0	6	0

Figure 2.1 a): Adjacency Matrix(directed)

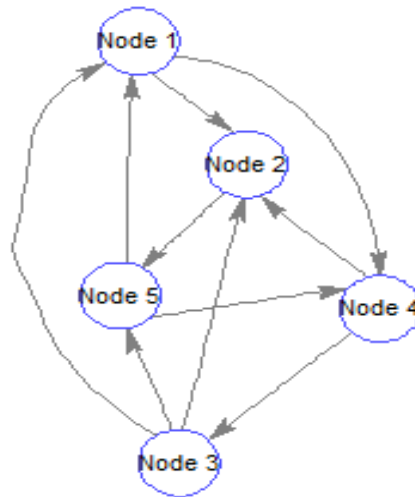


Figure 2.1 b): Directed Graph

For an undirected graph, entry  $a_{i,j} = a_{j,i}$ , that is, adjacency matrix is symmetric. We can see from figure 2.2 a) and b). For example, node 1 represent Stuttgart city and node 4 as Munich, traffic fee for through train between this two cities is 9 euros, no matter which one is the source.

$$A = \begin{bmatrix} 0 & 4 & 7 & 9 & 2 \\ 4 & 0 & 1 & 9 & 2 \\ 7 & 1 & 0 & 6 & 5 \\ 9 & 9 & 6 & 0 & 0 \\ 2 & 2 & 5 & 0 & 0 \end{bmatrix}$$

	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	0	4	7	9	2
Node 2	4	0	1	9	2
Node 3	7	1	0	6	5
Node 4	9	9	6	0	0
Node 5	2	2	5	0	0

Figure 2.2 a): Adjacency Matrix(undirected)

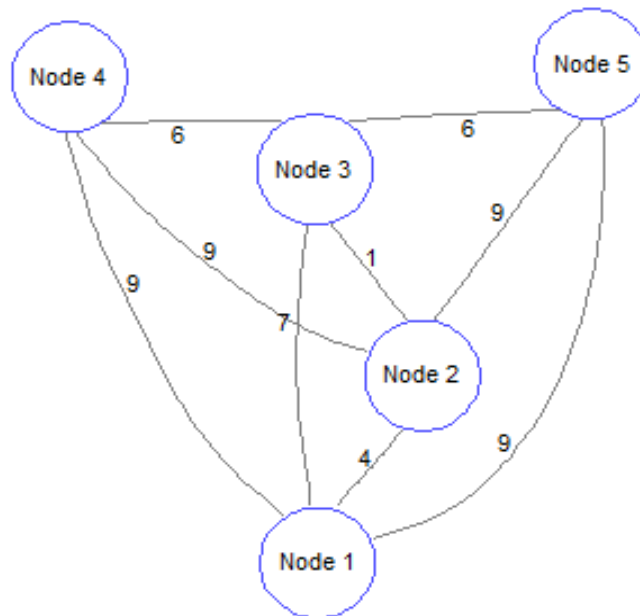


Figure 2.2 b): Undirected Graph

There`s a special case in graph layout, self-connection. If an edge`s two vertices is the same, we say this vertex is self-connected. In adjacency matrix  $A$ , if  $a_{i,i}$  is nonzero entry, means node  $i$  has link to itself. A graph without self-connection is named simple graph, in most case, graph is simple graph. We contribute to visualize connection between nodes, and more extra edges will make the layout graph visually confusing, in our node-link layout, put an 's' in the node id to represent self-connection to reduce edges. Refer to figure 2.3, Node 1,2,3,4 all contain self-connection.

In this thesis, we will concern more about directed graph because it can display more information than undirected graph. For example:

	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	1	1	0	1	0
Node 2	0	1	0	0	1
Node 3	1	1	1	0	1
Node 4	0	1	1	1	0
Node 5	1	0	0	1	0

Figure 2.3 a): adjacency matrix with self-connection

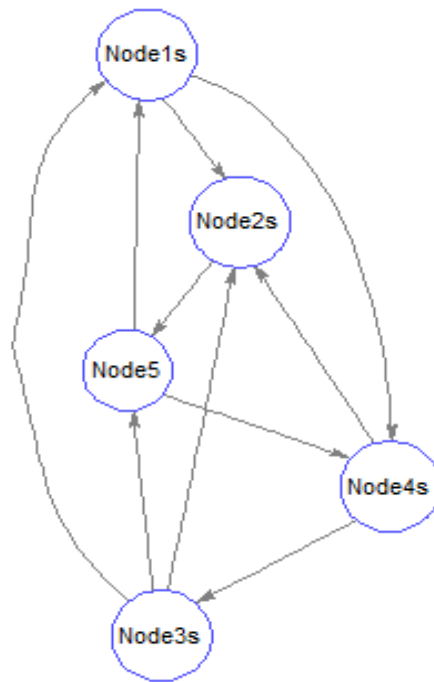


Figure 2.3 b): 's' in the node id to represent self-connection

## 2.3 Important Algorithms in Graph Theory

Graph theory is an vital subject or foundation of graph visualization. We could have in-depth knowledge of graph data and, algorithms to manipulate the graph data structure. Graph theory deal with various problems: subgraphs, graph coloring, route problems network flow and so on. In this thesis, we study route problems, highlight characteristics and reveal underlying meaning of graph data, implement these theory and apply them in graph visualization. Since adjacency matrix is the mathematical representation of a graph, graph theory is algorithms and methods about matrix. It's a mathematical procedure manipulate facilitate automatism in graph visualization, help visualizator to analyze and visualize. For example, depth first algorithm searching the adjacency matrix provides a new adjacency matrix which includes paths visiting order, a sequence of nodes in topological order.



### 2.3.1 All Pairs Shortest path

Finding shortest path is one of the most optimization problem, also a classic algorithm in graph research. It has been applied widely, like route selection, addressing and circuit routing arrangement, used as foundation to solve other most optimization problem, forecast and decision-making. For example, a road map with  $n$  cities, shortest path or cheapest expense travel from one city to another could be found.

Consider in mathematics aspect, large amount of most optimization problem or dynamic programming problems can be viewed equivalently to find shortest path in directed graph<sup>[8]</sup>. In graph theory, shortest path algorithm had been thorough solved.

Assign real number to each edge, recorded as  $W$  and named as edge's weight. Sum of all the constituent edges' weight from node  $i$  to node  $j$  is the weight of path  $p$  and, the minimum weight is shortest path distance between node  $i$  and node  $j$ . Finding shortest paths for all pairs of nodes result a matrix  $D$ , consider a graph with  $n$  nodes,  $D$  is a  $n \times n$  matrix, entry  $d_{ij}$  is distance (weight) of a shortest path from node  $i$  to node  $j$ .

We use a set of in-pathed nodes to represent  $p$ ,  $p = \langle node_1, node_2, \dots, node_k \rangle$ , the weight (distance) of path  $p$  is the sum of the weights of its constituent edges:

$$W(p) = \sum_1^{k-1} W(i, i + 1)$$

The shortest path distance from node  $i$  to node  $j$  is

$$d_{ij} = \begin{cases} 0, & \text{if } i = j \\ \min(W(p)), & \text{if shortest path } p \text{ from node } i \rightarrow j \text{ exists} \\ \infty (\text{inf}), & \text{otherwise} \end{cases}$$

In the Floyd-Warshall algorithm, 'intermediate' nodes is be introduced. A path  $p = \langle node_1, node_2, \dots, node_j \rangle$ , an intermediate node  $k$  is any node of  $p$  but not node 1 nor node  $j$ , that is, any node in the node set  $\{node_2, node_3, \dots, node_{j-1}\}$ .

Consider all  $n$  nodes of a graph be  $\{node_1, node_2, \dots, node_n\}$  and a subset  $\{node_1, node_2, \dots, node_k\}$ . Let  $d_{ij}^{(k)}$  the weight of a shortest path from node  $i$  to node  $j$  with all intermediate nodes in the node subset and  $p$  a shortest path among the subset. Floyd-Warshall algorithm is to explore the relationship between path  $p$  and shortest paths from node  $i$  to node  $j$  with intermediate nodes in the subset. The relationship depends on whether or not node  $k$  is an intermediate node of path  $p$ <sup>[9]</sup>.

If  $k=0$ , the shortest path is from node  $i$  to node  $j$  directly with no intermediate

node at all, has at most one edge, hence  $d_{ij}^{(0)} = W(i, j)$ .

If  $k \geq 1$ , intermediate nodes of path  $p$  exist, then splits  $p$  into a subpath from node  $i$  to node  $k$  and node  $k$  to node  $j$  as figure 2.4. Node  $k$  is not an intermediate node in node  $p_1$ , so  $p_1$  is the shortest path from node  $i$  to node  $k$  with intermediate nodes in  $\{\text{node } 1, \text{node } 2, \dots, \text{node } k-1\}$ . The same hold for  $p_2$  the shortest path from node  $k$  to node  $j$  with intermediate nodes in  $\{\text{node } 1, \text{node } 2, \dots, \text{node } k-1\}$ .

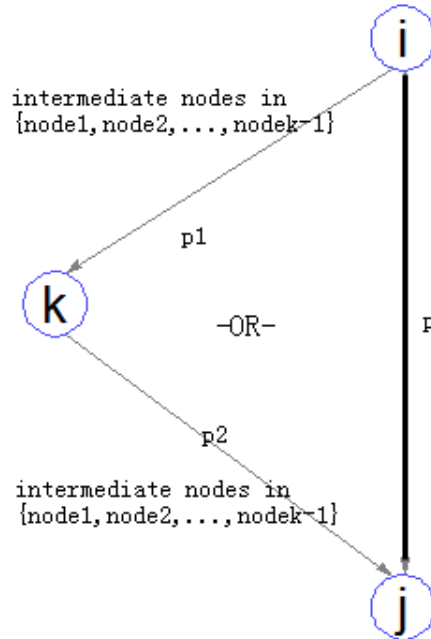


Figure 2.4:  $k=0$ , path  $p$  is a shortest path from node  $i$  to node  $j$ ; otherwise, intermediate node exist, break  $p$  into  $i \xrightarrow{p_1} k \xrightarrow{p_2} j$

Based on above structure, the following is a recursive solution to all pairs shortest path:

$$d_{ij}^k = \begin{cases} W(i, j) & , \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & , \text{if } k \geq 1 \end{cases}$$

For all pairs of nodes result a distance matrix  $D^{(n)} = d_{ij}^{(n)}$ . A bottom up procedure can be derived from above recurrence, it's input is an  $n \times n$  adjacency matrix and output  $D^{(n)}$ .

#### FLOYD-WARSHALL (W)

```

1  n ← rows[W]
2  D(0) ← W
3  for k ← 1 to n
4      do for i ← 1 to n
5          do for j ← 1 to n
6              do dij(k) ← min(dij(k-1), dik(k-1) + dkj(k-1))
7  return D(n)

```

The running time of Floyd-Warshall algorithm is  $\theta(n^3)$ , three variables i, j, k in the nested loops, each of the variables can take n possible values.

Take an example of the matrix details to have better understand of Floyd-Warshall.

For input  $W = D^0 = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 12 & 5 \\ 4 & 0 & 0 & -1 \\ 2 & -4 & 0 & 0 \end{bmatrix}$ ,  $n=4$ , obtain the follow  $D^{(n)}$  when recurrence algorithm executed<sup>[10]</sup>.

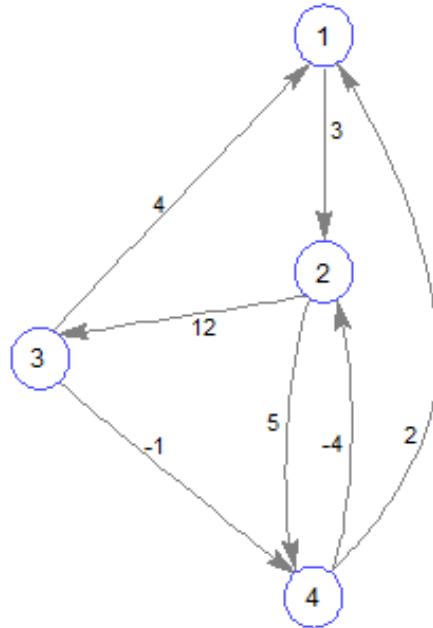


Figure 2.5: Example of Floyd-Warshall

$$D^1 = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 0 & 0 \end{bmatrix}$$

$$D^3 = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 7 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

$$D^2 = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 0 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

$$D^4 = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 7 & 0 & 12 & 5 \\ 1 & -5 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

### 2.3.2 Depth First Search Algorithm (DFS)

Depth First Search Algorithm(DFS) is the primary algorithm in graph theory, many other algorithms are more or less affected by DFS. DFS is a kind of graph traversal algorithm that traversal each component of a graph, it plays a crucial role in nodes positioning, graph partitioning and so on<sup>[11]</sup>.

DFS originates from Maze solving problem, begin with an entrance, walk every not repeated corridor and you will find a way out at last. This maze walking arrangement can be considered as a graph, corridors as edges while view intersections, corners and dead ends as nodes. The DFS idea is developed base on this maze solving principle, start from entrance, have to search all corridors, go out from the entrance. Mark corridors you have been searched to avoid repeated search, find and walk along the unsearched corridors as deeply as you can when reach an corner. At certain location, way back if you couldn't find unmarked corridor or meet an dead end. Repeat and repeat, all corridors could be searched and go out the maze from entrance<sup>[12][16]</sup>.

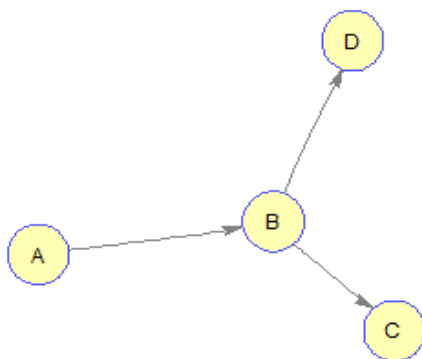
According to above theory, the process of DFS algorithm is:

1. Mark all edges and nodes 'unsearched'/'unvisited' and start search from random node;
2. Choose an 'unsearched' edge and visit connected node, mark the edge 'searched';
3. Repeat step 2 until reach the node which has is 'unsearched' edge , way back to previous visited node and go back to step 2;
4. If way back to started node, choose an unvisited node and go to step 2;
5. End search if all nodes and edges are searched.

Let`s take an example to explain DFS. Given an adjacency matrix

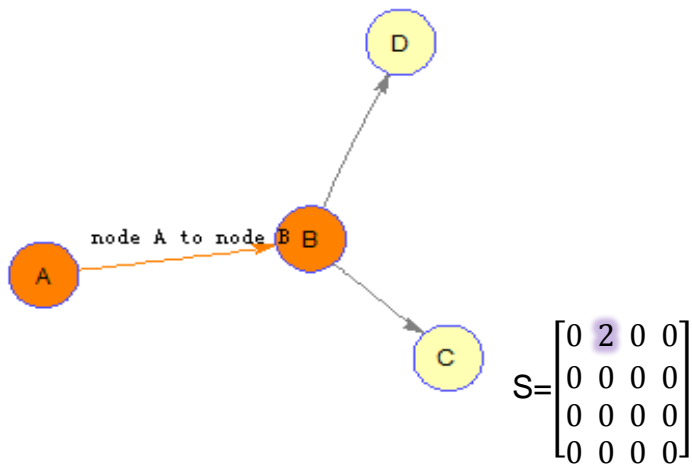
$$G = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Have corresponding graph:

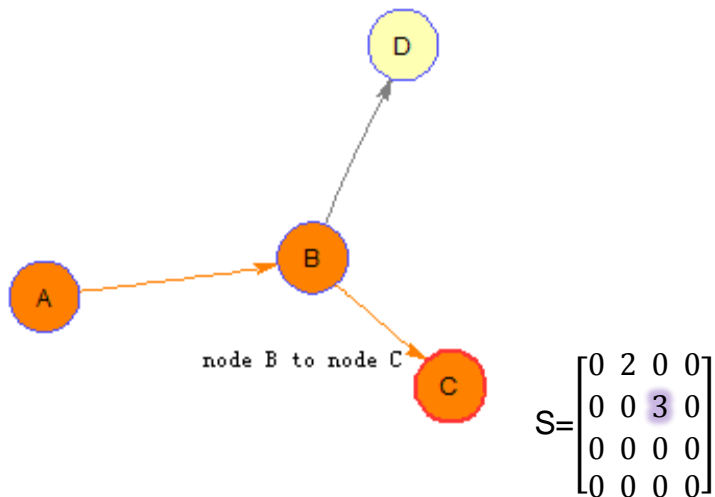


We use another adjacency matrix S which to record the node visiting order, visit in ascending order. S has the same size as graph adjacency matrix G, entry  $S_{ij}$  means visit node j from node i.

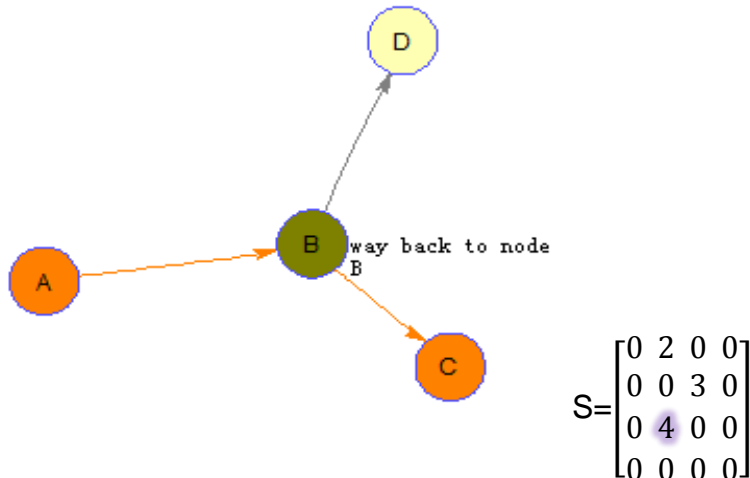
Begin from node A for example, walk along unsearched edge to node B:



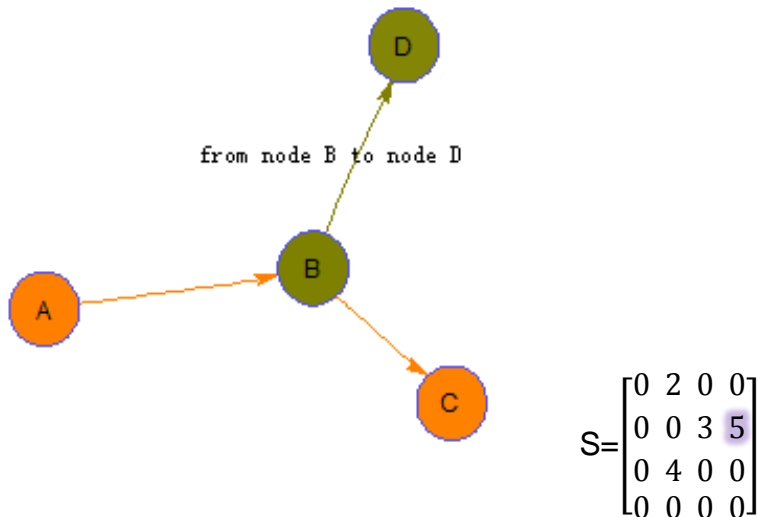
At node B, pick up an unsearched edge and go to next node:



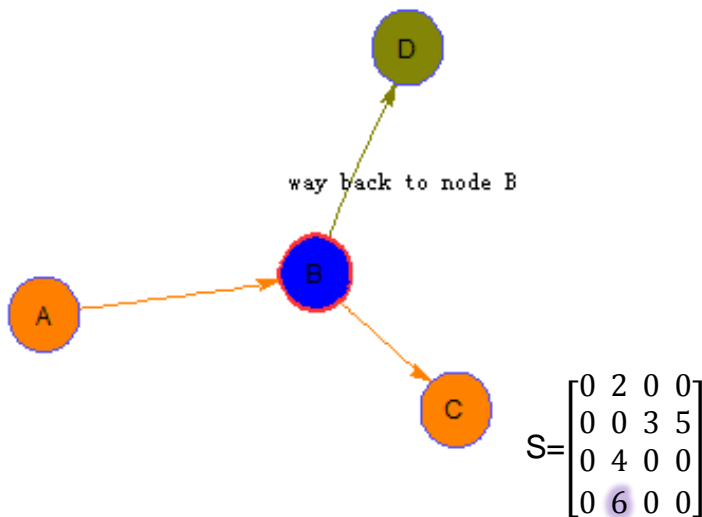
From node C, couldn't find any unsearched edge that connected to node C, so way back to previous Node B:



From node B, we can find unsearched edge, along this edge to node D:



There is no unsearched edge that connected to node D, so way back to its previous node B:



The same, back to started node A (entrance), all edges and nodes are searched, end the search process. We have the final matrix S as follow:

$$S = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 7 & 0 & 3 & 5 \\ 0 & 4 & 0 & 0 \\ 0 & 6 & 0 & 0 \end{bmatrix}$$

Besides, two additional sequence be provided to record all nodes (sequence k) and each node's corresponding previous node(sequence f):

$$k = A B C D$$

$$f = 0 A B B$$

DFS returns an topological order of nodes after traversing a graph. If each node means an event, we could find out or layout the sequence order of all events and make reasonable schedule. At above example, resulted sequences k and f represent nodes' topological order, graph can be divided into three layers according to the order, node A in first layer, node B in second

layer, node C and D in third. When there is no particular request, DFS plays an important role in node positioning algorithm to improve graph layout aesthetics, it's resulted nodes topological order has great helpful in making the node-link graph drawn in more reasonable way. Although this node positioning algorithm won't be studied in the thesis, it still make sense for future work or study.

## Chapter 3 Graph Visualization/Graph Drawing

Graph drawing techniques will be discussed in this chapter since basic knowledge of graph and its mathematical operation are introduced. There are two major research areas: how to draw a pleasing graph and how graph reveal underlying information from input data.

### 3.1 Graph Drawing and Aesthetics

Why draw graph to represent information? One important reason is people have the need to visualize information. In other words, one function of graph drawing is to reveal relationship between objects. One example in our daily life is the subway map. When refer to a map to look for routes from one site to another or to multiple destinations, the following questions should be answered by the map: any station at the sites? which train lines pass one station? how to transfer if no nonstop train?

Obviously, use graph to represent these information is better than language. We can use a series of language to represent which stations each train line pass, like train 1 pass station A, B, C, train 2 pass station C, D, E, F, train 3 pass station E, G, H. A large amount of language to represent a little information, and it's not easy to extract any further more information. It's not difficult to find out the routes from station A to H if just three trains and eight stations, but rather confused when consider more and more trains and stations. It's convenient and intuitive using graph to visualize information.

Figure 3.1 is a node-link subway map of Hong Kong, each station on the map can be represented as node, train lines connect these nodes is edges. Edges in different color to distinguish different trains, put nodes at exactly real location. Figure 3.2 is a S-Bahn (suburban) subway map of Stuttgart, color coding edges to represent train lines, white dots on the edges is approximately location which mapping the real map, and use bigger nodes to represent two ends of train lines. Further more, all urban subways in figure 3.2 are drawn in gray line, not affect the whole graph but enrich graph information. Figure 3.1 combine subway node-link graph with real geographical city map make it has two functions: a subway map and city map as well. Almost all important information that we need are layout on these graphs even they layout in different ways. You can not reveal all information and consider every aspect on a graph of course, it depends on specific task. What important is the graph visualization technique could provide explicit relationship between objects, and identify pairs of connected nodes is precisely what we need in this thesis.



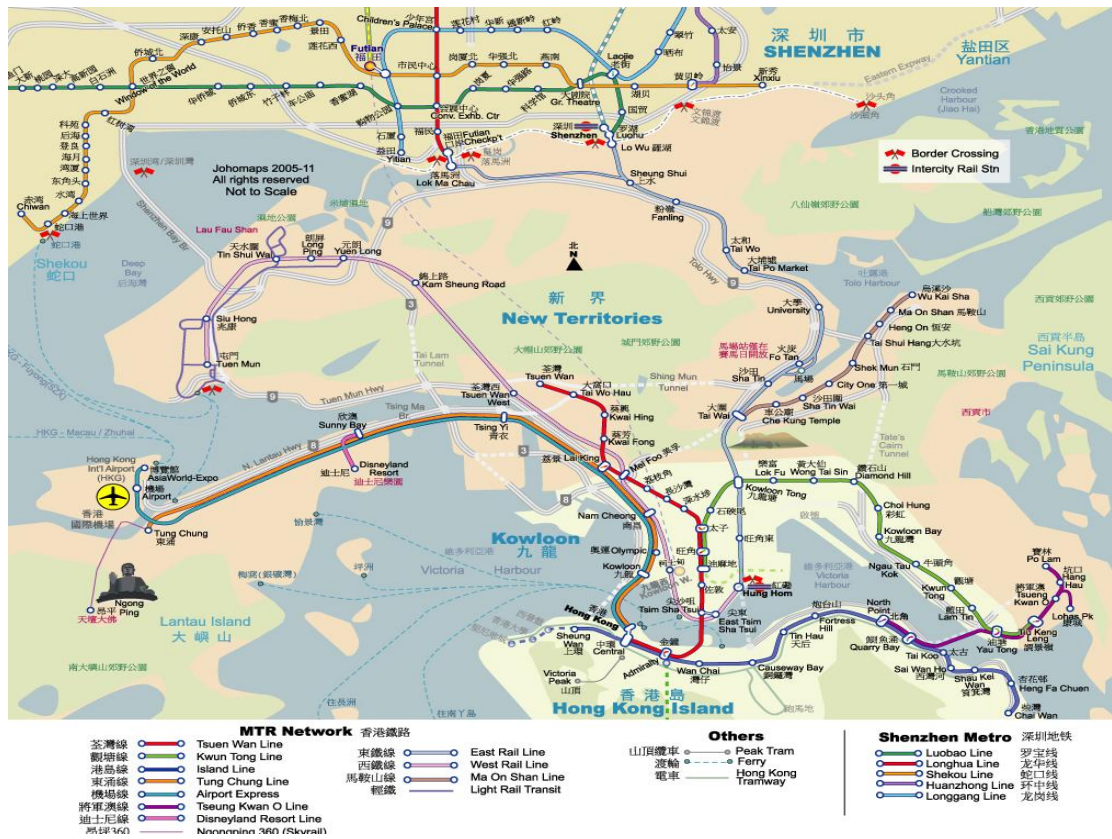


Figure 3.1 Subway map of HongKong

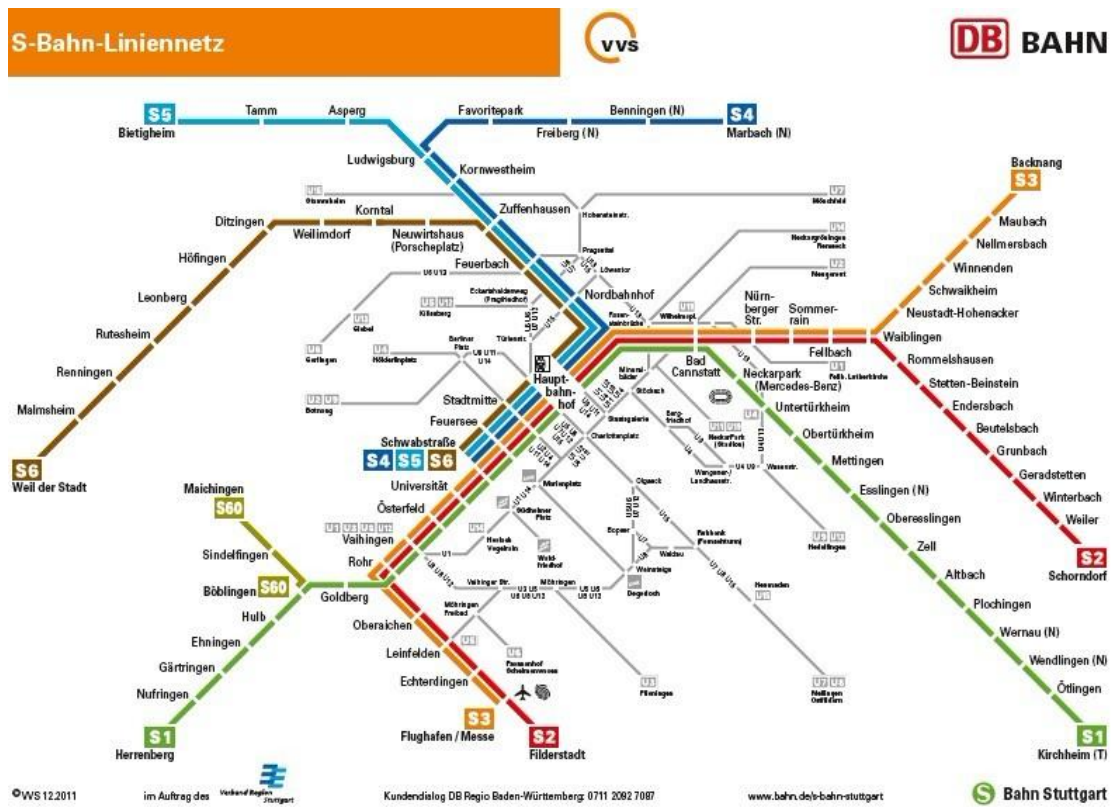


Figure 3.2: S-Bahn map of Stuttgart

A 'bad' node-link graph normally suffer from three fundamental problems:

(1) Graph become cluttered and indecipherable, normally because nodes getting too close or even overlap, too much edges crossing or edges drawn in inappropriate way.

(2) Nodes are placed in wrong position. Nodes position affect the understandable and readable of graph very much, not just about graph aesthetics but also graph meaning representation, graph could be interpreted different when draw nodes in different positions even with the same input<sup>[24]</sup>.

(3) Violate perceptual custom. For example, strongly or highly related nodes drawn far away from each other or connected by long links, irrelevant nodes positioned closely on the contrary.

It's difficult to define a graph is 'nice' or 'bad' precisely, but there are still some commonly accepted criteria or principles for graph drawing. The difficulty to lay out a readable and understandable graph can be summarize to two parts<sup>[14][16][17]</sup>.

1. Syntactic. Mainly about graph aesthetics, how to lay out a beautiful graph, avoid overlap between nodes or edges.
2. Semantic. How a graph reveal information: task specific, underlying meaning, data characters.

One important reference is Norman's three levels model. Norman group the process which viewer extract edges shape and information into three levels: visceral, behavioural and reflective. This 'process principles' can be applied to graph drawing help us to evaluate graph visualization.

Visceral level is about perception and impression of objects. One important principle which we will apply to hierarchical clustering is grouping principles: elements could be grouped according to similarity, continuation, proximity connectedness or familiarity. Behavioural level is affected by visceral level result, focus on graph's usability and readability. Which means: our cognition of graph is influenced by aesthetics. Understanding the results from previous two processes is reflective level. How we find the meaning of a graph is influenced by culture, education, environment and so on. Figure 3.1 place nodes(stations) on real location of map, use it's background (real Hong Kong map) convenient to tourist with different background, whereas figure 3.2 enhance the shape of node-link graph.

For node-link graph drawing, Purchase present 'metrics for seven aesthetics'<sup>[15]</sup> about nodes and edges placement:

- minimizing edge crossings,
- minimizing edge bends,
- maximizing symmetry,
- maximizing the minimum angle between edges leaving a node,
- maximizing edge orthogonality,
- maximizing node orthogonality,
- maximizing consistent flow direction (directed graphs only).

It's difficult to apply all these metrics and aesthetics criteria simultaneously.

For example, conflict between symmetry and grouping nodes in clustering algorithm. Figure 3.3 illustrates conflict minimizing edge crossings and maximizing symmetry for the same adjacency matrix input.

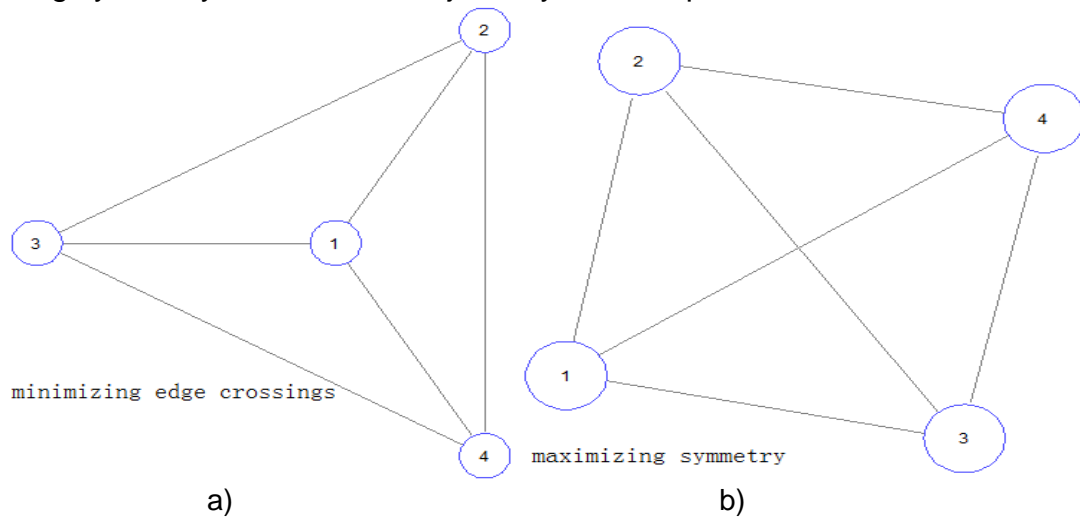


Figure 3.3: a) minimizing edge crossing layout

b) maximizing symmetry layout

### 3.2 Force-directed Layout Algorithm

Force-directed is popular because the algorithm is easy to understand and program, it encodes aesthetic criteria, results visually pleasing graph, do well at distributing nodes evenly, minimizing edge bends and maximizing symmetry.

#### 3.2.1 Spring embedder (Eades, Kamada and Kawai)

Force-directed algorithms view graph as system of bodies, task is about the configuration of forces between bodies let the system with locally minimal energy. Let's take a look at an example about the force here. Eades<sup>[18][21][23]</sup> present an concept of spring embedded, replace edges by springs, calculate forces between nodes and edges. The resulted force will let the spring stay at an certain length, attractive force arise when spring is stretched and repulsive force push nodes away if they get too close.

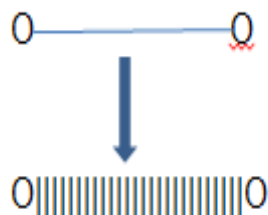


Figure 3.4: replace edge with spring

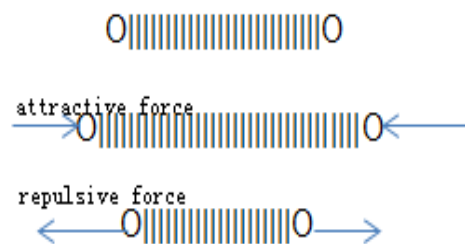


Figure 3.5: force between nodes

Replace all edges by spring, calculate forces to position nodes and config length of edges, system will reach an equilibrium state: sum of forces on each body is zero. This state defines a kind of straight line drawing, rough configuration process schematic diagram in Figure 3.6.

Although the physical system of springs is modeled, Hooke's law\* is not implemented in algorithm. It's actually defined as energy system rather a force system, algorithm config system to reach locally minimal energy state. A variety of force formulas had been developed, all use the concept of 'force' rather than realistic one. Eades define the attractive and repulsive forces as:

$$f_a = k_a \log d$$

and

$$f_r = k/d^2$$

where k is the optimal and d is the initial distance between nodes.

Kamada and Kawai<sup>[20]</sup> used another algorithm which is based on spring graph system: for a pairwise connected nodes, force try to keep them in ideal distance which is proportional to their graph theoretical shortest path. They define energy of the graph as:

$$\sum_{i < j \leq |V|} k_{ij} (|p_i - p_j| - l_{ij})^2$$

Where  $p_i$  is the position of the spring corresponding to node i,  $k_{ij}$  is the spring constant between  $p_i$  and  $p_j$ , and  $l_{ij}$  is the ideal distance between node i and j. By solving partial differential equation, energy is reduced until it below a preset threshold and node is reposition during this process.

---

\*Hooke's law<sup>[19]</sup>:

The concept of Hooke's Law is that the amount of force applied to a spring or elastic object is proportional to the amount of deformation (length of stretch or compression). The greater the force applied to an elastic object, the more deformation (stretch or compression) there is. With less force applied, there will be less deformation in the spring. The formula for Hooke's Law is  $F_x = -k(x - x_0) = -kx$ . Where  $k$  is the force constant of the spring,  $x$  is the amount of deformation in meters.

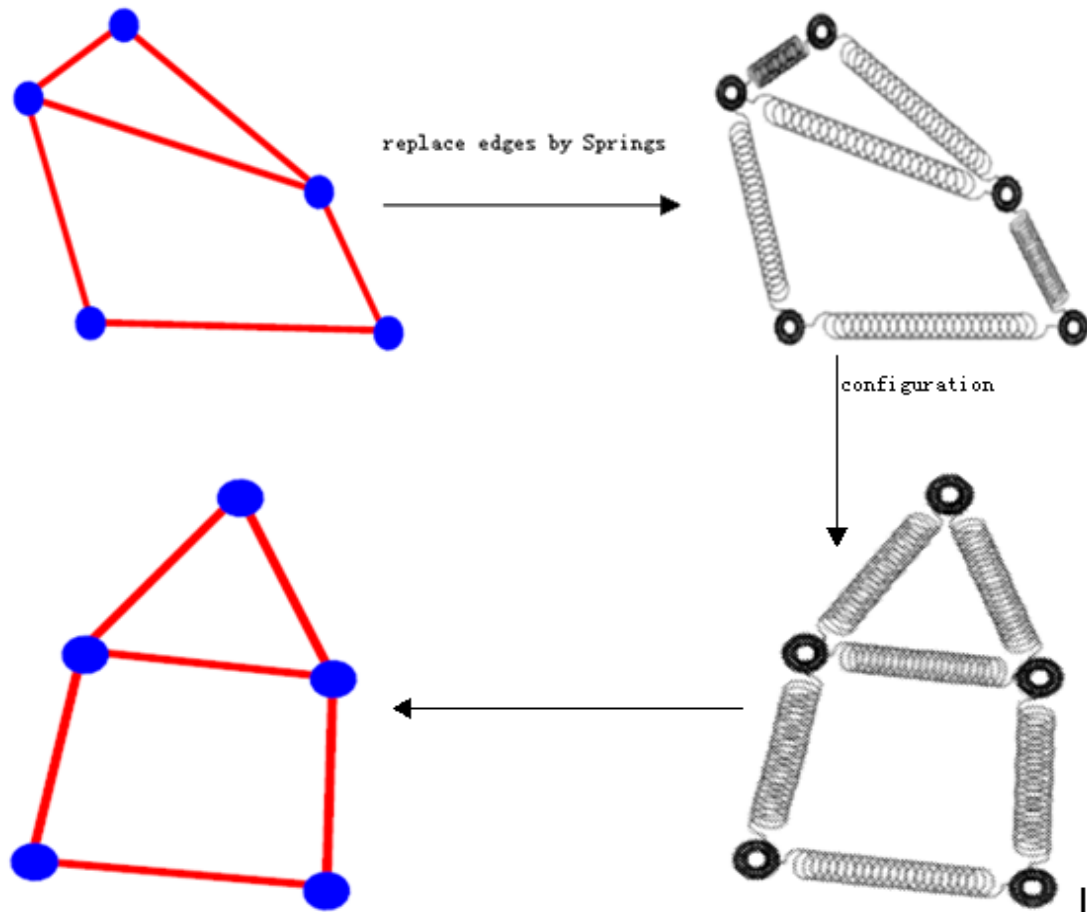


Figure 3.6: Spring Algorithm (Eades)

### 3.2.2 Force-directed Method (Fruchterman and Reingold 90)

What special in Fruchterman algorithm are: control the graph drawing with a square wall boundary to prevent nodes and edges outside the wall, and prevent nodes be drawn too close to each other, attractive forces only be calculated between pairwise connected nodes, but repulsive forces are considered between all nodes just like Eades.

Initial state is not or partly considered in this algorithm, normally nodes are placed randomly at first. Iteration number is chose by user or programmer, the effect of attractive forces between pairwise connected nodes and repulsive forces between all nodes be calculated in each iteration, a 'cool down' anneal to limit displacement finally. During the iteration, a restriction condition be introduced to keep nodes` new position still within user-defined square boundary. Pseudo-code is given in Figure 3.8.

(a) Forces calculation

The area of frame is predefined as a rectangle:  $W * L$ , where  $W$  is the width and  $L$  as length, nodes are positioned randomly in this frame first, and let  $d$  the initial distance between two nodes.

The optimal distance between nodes is calculated as:

$$k = C \sqrt{\frac{\text{area of frame}}{\text{number of nodes}}}$$

where C is experimental based constant, it's value better be chose between 0.5 and 0.7 according to Dr. Scott White from University of California, Irvine. Our goal is to place nodes in the frame evenly, apart two nodes away if they get too close or push them closer if nodes are too far away from each other. Let  $f_a$  and  $f_r$  be the attractive and repulsive forces respectively, the forces formulas are defined as:

and

$$\begin{aligned} f_a &= d^2/k \\ f_r &= k^2/d \end{aligned}$$

after some experiments. The sum of  $f_a$  and  $f_r$  equal to zero under certain distance value, this value is k the ideal distance between nodes. In other words,  $f_a$  and  $f_r$  counteract each other when nodes place at ideal distance, illustrated in Figure 3.7.

Since the initial nodes position is random, a special case exists: nodes are overlap or in the same position. This algorithm predefine a constant to replace  $d$  if above situation happens. The value of this constant should not larger than frame area of course.

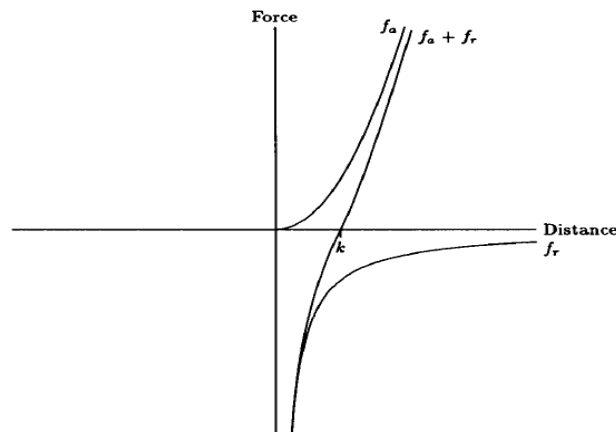


Figure 3.7: forces versus distance<sup>[4]</sup>

Pseudo code for Fruchterman Force-directed Algorithm<sup>[4]</sup>:

```

area := W * L; { W and L are the width and length of the frame }
G := (V, E); { the vertices are assigned random initial positions }
k :=  $\sqrt{\text{area}/|V|}$ ;
function  $f_a(z)$  := begin return  $x^2/k$  end;
function  $f_r(z)$  := begin return  $k^2/z$  end;

for i := 1 to iterations do begin
  { calculate repulsive forces }
  for v in V do begin
    { each vertex has two vectors: .pos and .disp }
    v.disp := 0;
    for u in V do
      if (u # v) then begin
        {  $\Delta$  is short hand for the difference }
        { vector between the positions of the two vertices }
         $\Delta := v.pos - u.pos$ ;
        v.disp := v.disp + ( $\Delta / |\Delta|$ ) *  $f_r(|\Delta|)$ 
      end
    end
  end
  { calculate attractive forces }
  for e in E do begin
    { each edge is an ordered pair of vertices .v and .u }
     $\Delta := e.v.pos - e.u.pos$ 
    e.v.disp := e.v.disp - ( $\Delta / |\Delta|$ ) *  $f_a(|\Delta|)$ ;
    e.u.disp := e.u.disp + ( $\Delta / |\Delta|$ ) *  $f_a(|\Delta|)$ 
  end
  end
  { limit the maximum displacement to the temperature t }
  { and then prevent from being displaced outside frame }
  for v in V do begin
    v.pos := v.pos + (v.disp / |v.disp|) * min(v.disp, t);
    v.pos.x := min(W/2, max(-W/2, v.pos.x));
    v.pos.y := min(L/2, max(-L/2, v.pos.y))
  end
  end
  { reduce the temperature as the layout approaches a better configuration }
  t := cool(t)
end

```

Figure 3.8: Force-directed placement

### (b) Drawing boundary

In this algorithm, drawing region is modeled as an static, ‘four walls’ object, each wall with normal force ‘exactly equal to the force pushing any node beyond it to stop the node ‘like a real wall’<sup>[4]</sup>. All walls of this region are orthogonal to coordinate system to make it easier to implement.

If two nodes are far away enough, the repulsive force between them does not contribute much to displacement, such repulsive forces are omitted when calculating sum of repulsive forces. This is so called grid boxes technique. Grid boxes used as threshold, nodes which is in distance far way beyond the threshold be ignored<sup>[22]</sup>. Only repulsive forces between nodes lying in nearby boxes be considered. See Figure 3.9. Another refinement is to model the border after physical frame. The length of border is compute according to displacement vectors, move nodes to a restricted area so that the resulted graph never near the border. Figure 3.10 is the modification along x-coordinate, the

same thing happened to nodes` y-coordinate.

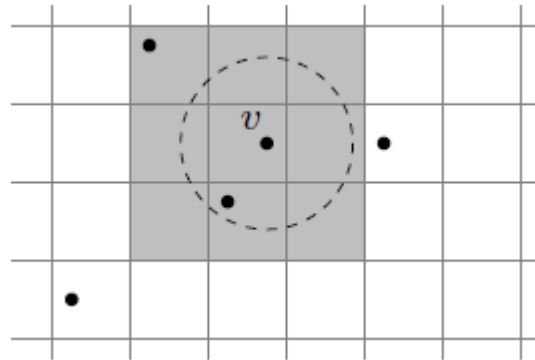


Figure 3.9: Grid boxes technique

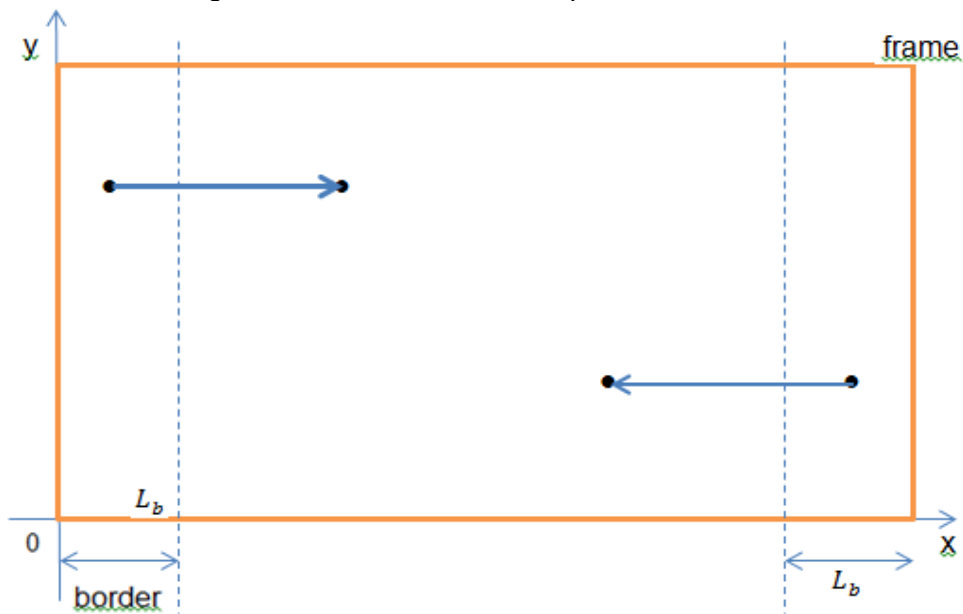


Figure 3.10: modeling the frame along x-coordinate

Finally, an important simulation 'cool down' is considered. Temperature  $T$  is set to limit maximum displacement of nodes, the temperature and maximum displacement will be changed at each iteration. A good cool down schedule is help to have better graph layout. We define  $T$  as frame area related,  $T = W/10$  for example. Experimentations tell us that low temperature  $T$  leads to better and fewer iterations (according to <sup>[4]</sup> and Dr. Scott White).



### 3.3 Partial Drawn Links in Directed Graph (Edges Shorten)

In node-link graph, objects represent as nodes and links as relationship between objects as introduced. Viewer's usual habits of observation is from initial node, find out node which is at the other end of the edge by following the link (descendant). It's an important task for visualizator to looking for node's descendants. For example, Figure 3.1 and 3.2 the subway graphs, from departure station, locate terminal station by following the subway line. Another example is an email record system as illustrated in Figure 3.11, want to know David email to whom, or all people David ever contacted. From the node which represents David, along all links send out from the node, go to targets at the other ends of these links.

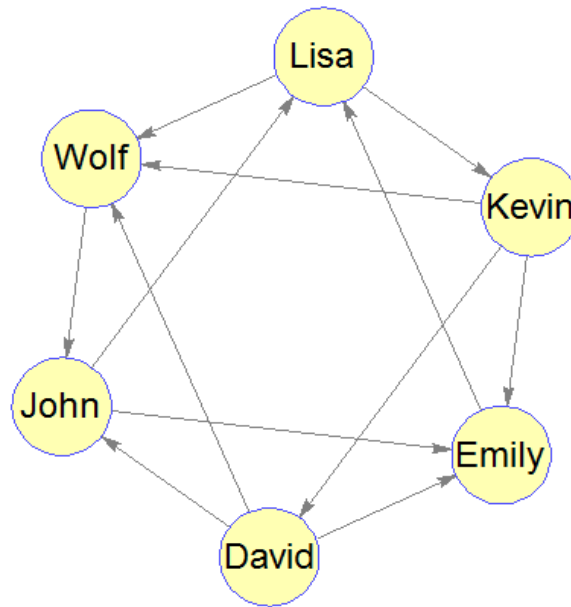


Figure 3.11: Graph of email record

#### 3.3.1 Type of Edges

Links or edges play a media role in finding descendant nodes, thus, algorithms deal with edge drawing is important. There are three types of usually used edges: curve, polyline and straight line. One of straight line drawing: force-directed has been introduced, let's have an rough overview of other two types but not go to drawing algorithm details.

##### (a) Curve

Curved edges is one of drawing conventions, edge crossings can be reduced, avoid nodes and edges get too close, enhance graph readability by using curved edges. Figure 3.12 a) and b) are drawn from same data input and place nodes at same position, edge crossing is avoided when replace one straight line edge by curve. In Figure 3.13 a), edge from node A to node C not just go through node B but almost parallels other two edges, these two problems is

solved in b) perfectly. A variety of curve edge drawing algorithms has been developed, one instance is Bezier curves<sup>[26]</sup>, which use calculated smooth curve to avoid edge crossing through nodes and graph aesthetics is improved.

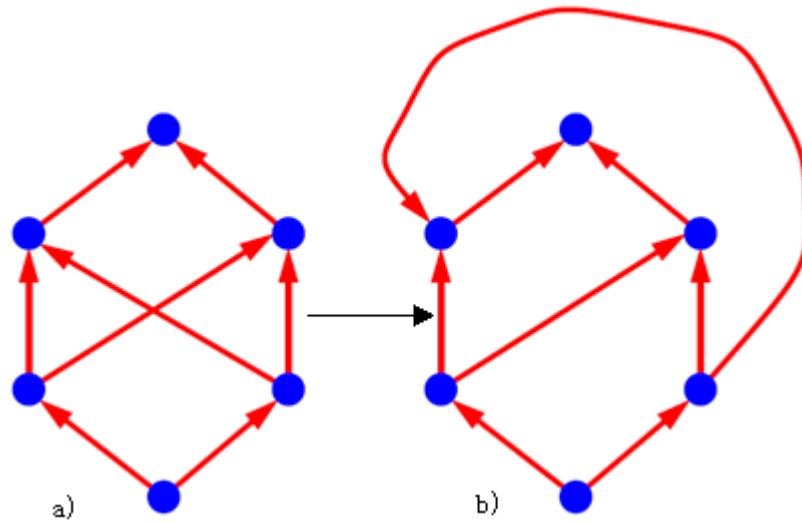


Figure 3.12: a) all straight edges  
b) curved edge instead

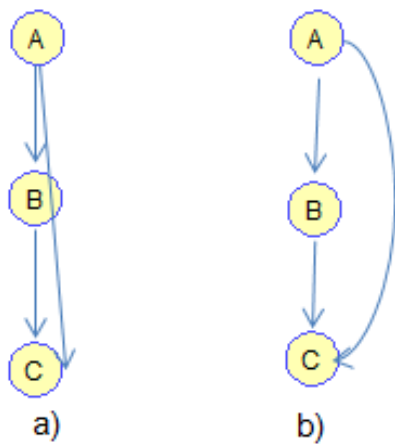


Figure 3.13: a) all straight edges;  
b) with curved edge

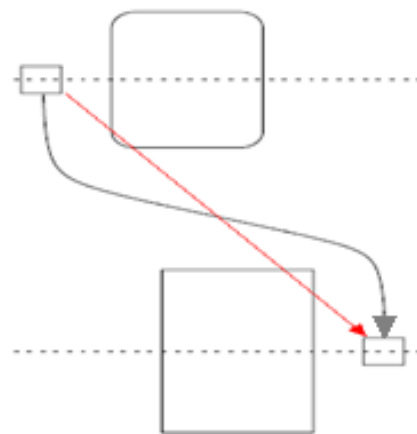


Figure 3.14: Bezier curve

(b) Polyline (bended edges)

Polyline is used in orthogonal grid drawing: edge consists of segment lines, or in other words, bend straight line with at arbitrary angle, and these segment lines are parallel with coordinates axis. Figure 3.15 is an example of orthogonal drawing, and figure 3.16 is an upward grid drawing with area-efficient which presented by Garg Goodrich Tamassia<sup>[27]</sup>. Graph is well structured and has good visual: edges not too close to each other, edges bended angle is uniform.

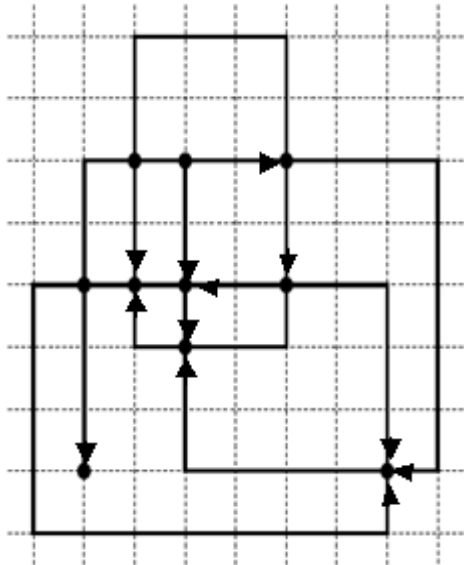


Figure 3.15: Orthogonal grid drawing

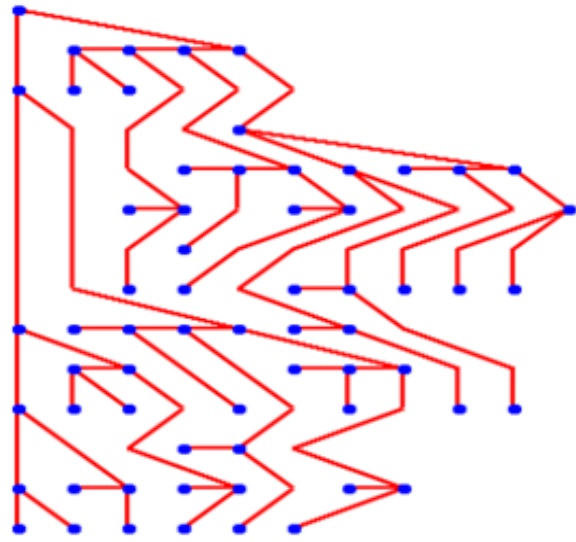


Figure 3.16: polyline upward grid drawing

Compare above edges drawing, the turning point at bended edge could cause visual confusion, create visual illusion that the turning point is a node. Curve conforms with visual habits much more better, viewer can follow the edge to find the other end under the orientation of arrow more 'smoothly', while bended edge weaken the orientation function of arrowed edge.

Straight edge has best orientation effect within these types, guiding human visual direction quite well. A daily life example is the street signs, a straight line with arrow head to indicate destination direction, see Figure 3.17 a). The end of the link with arrow doesn't connect to the target, but couldn't prevent us to find the destination under the arrow's guiding. The corresponding node-link graph representation is drawn in Figure 3.17 b), use partially drawn straight link shoot out from one node to find out it's descendants while the end with arrow do not need to have contact with target nodes.

A new node-link graph drawing technique will be introduced in following section: use partially drawn straight links to reduce edge crossings, minimizing edge bends, enhance graph readability and convenient for viewer to find out descendants. This thought is put forward by Dr. Burch from Stuttgart University and developed under his supervise.

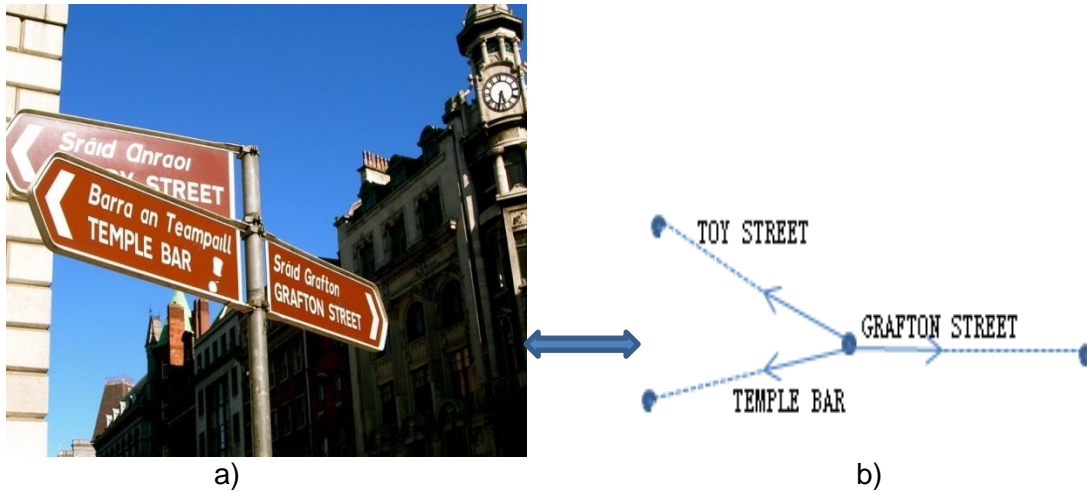


Figure 3.17: a) street sign at Dublin  
 b) graph representation

### 3.3.2 Partially Drawn Links Methods

In partially drawn links technique, straight line is used to draw edges, start and end points are lay on a line. The first part of the link is like other techniques, a starting node, link shoot out from this node, then the first question we have to answer is where the other end (with arrow head) place to. We put the drawing frame at the quadrant I of x y coordinates, nodes on the link can be calculated as their coordinate point. Then the first question be solved as:

1. for each pairwise nodes, derive linear equation from start and end nodes` coordinate point;
2. base on start node`s coordinate point and linear equation, control the link`s length according to a given scaling parameter or user-defined.

#### (a) Method 1

Do not consider orientation relationship between start and end nodes first, take a pair of connected nodes, their x y coordinate relationship is illustrated in Figure 3.18 a). The start node is static and its position is given, we introduce a intermedia node (see Figure 3.18 b) which also lay on the link to control the arrow head position, then the length of the link could be partially drawn.

Substitute start node  $(x_1, y_1)$  and ended node  $(x_2, y_2)$  coordinate points to linear equation  $y = kx + b$ :

$$\begin{cases} y_1 = kx_1 + b \\ y_2 = kx_2 + b \end{cases}$$

Solve equation set to get  $k$  and  $b$ .  $b$  is the coordinate point where link intersects x-coordinate when  $y = 0$ .  $k$  is slope of the link, which is  $k = \tan \theta = (y_2 - y_1)/(x_2 - x_1)$ . Value range of  $x_3$  is  $x_1$  to  $x_2$ . We introduce a parameter  $a$  to control value of  $x_3$  that enable the intermedia node could be any point

in the link, which is:

$$x_3 = \frac{(x_2 - x_1)}{a},$$

substitute  $x_3$ ,  $k$  and  $b$  to linear equation to get

$$y_3 = \frac{k(x_2 - x_1)}{a} + b$$

Then we get the end node's coordinate point:

$$(x_3, y_3) = \left( \frac{(x_2 - x_1)}{a}, \frac{k(x_2 - x_1)}{a} + b \right)$$

This coordinate point is used to control the arrow head position which is the length of link.

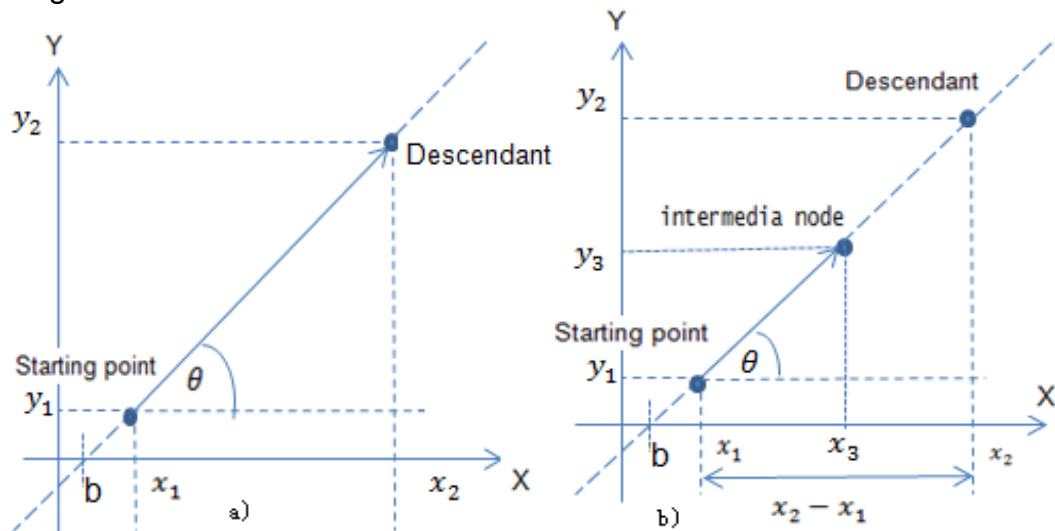


Figure 3.18: a) x-y coordinate of link

b) intermedia node introduced

For the coordinate point of intermedia node, we just need to confirm the value of  $x_3$  or  $y_3$ , the other one can be calculated. From the above derivation, have to choose from  $x_3 = (x_2 - x_1)/a$  or  $y_3 = (y_2 - y_1)/a$  first for position calculation. Assume  $a$  is fixed value, then we could have better control of arrow head for the numerator which has larger value. Because we just need to change the value a little bit, the length of link will have obvious change. It's more obviously when there are more edges. Another reason is we get result with small number for a numerator with small value, even need to precise to several digits after the decimal point. Computation time and difficulty is increased, higher demand for processor.

There are four situation four location relationships between start and end nodes, illustrated in Figure 3.19, which the end node at the direction of northwest, southwest, southeast and northeast respectively. Because the coordinate point of start and end nodes are known, means slope of link is known, then the direction relationship is become whether  $x_2$  or  $y_2$  is larger or smaller than  $x_1$  or  $y_1$ . Then whole process can be summarize as:

1. Get linear equation form coordinate point of start and end nodes;
2. Pick up the larger absolute value different from  $x_2 - x_1$  or  $y_2 - y_1$ . Use x-coordinate for example, one situation is  $x_2 \geq x_1$ , then  $x_3 = (x_2 - x_1)/a$ , the other situation is  $x_1 \geq x_2$ , then  $x_3 = (x_1 - x_2)/a$ . It's the same for y-coordinate.

The coordinate point of intermedia node is confirmed, length of link can be controlled by changing the value of a.

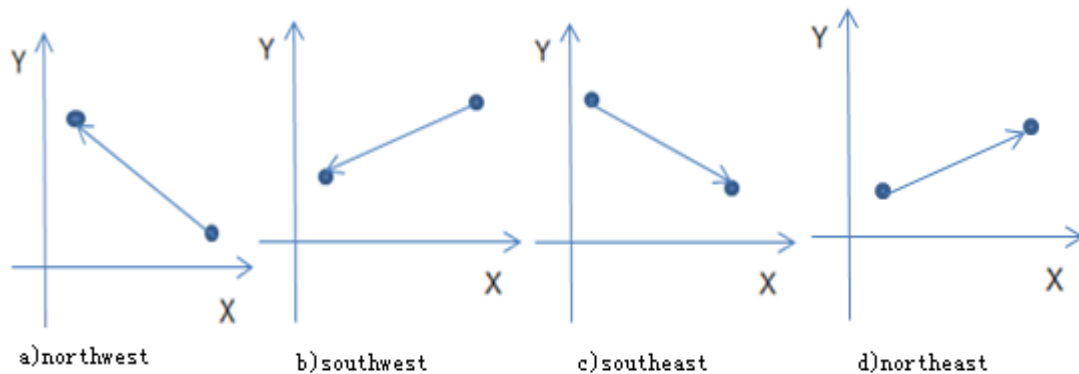


Figure 3.19: location relationship between start node and end node

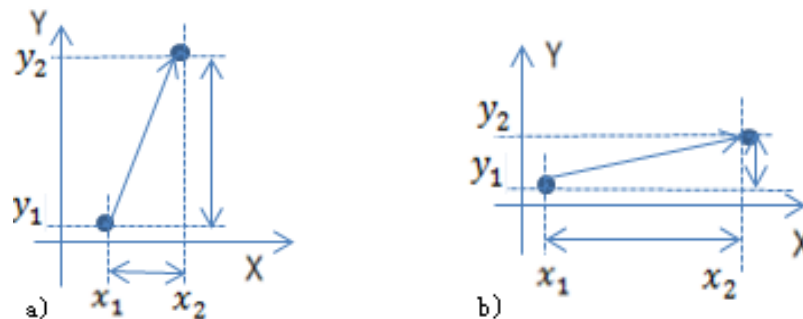


Figure 3.20: a)  $|y_2 - y_1| \geq |x_2 - x_1|$  . b)  $|x_2 - x_1| > |y_2 - y_1|$

### (b) Method 2

It's more understandable and readable if viewer could control length with a flat rate, for example: shorten link to its 50% length. This method is more acceptable by viewer. To achieve this, based on method 1, just need one more length control step in each intermedia node coordinate calculation. Here are the calculation steps:

1. Get linear equation form coordinate point of start and end nodes, and get link's length  $L = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$ ;
2. Control the length of link by intermedia node  $(x_3, y_3)$ , 50% length for example, just need to solve equation set:

$$\begin{cases} L * 50\% = \sqrt{(y_3 - y_1)^2 + (x_3 - x_1)^2} \\ y_3 = k * x_3 + b \end{cases}$$

With constrain conditions:  $(x_3, y_3)$  must between  $(x_1, y_1)$  and  $(x_2, y_2)$ . For viewer, it's intuitive method but higher running time than method 1. Figure 3.21 is the scheme diagram.

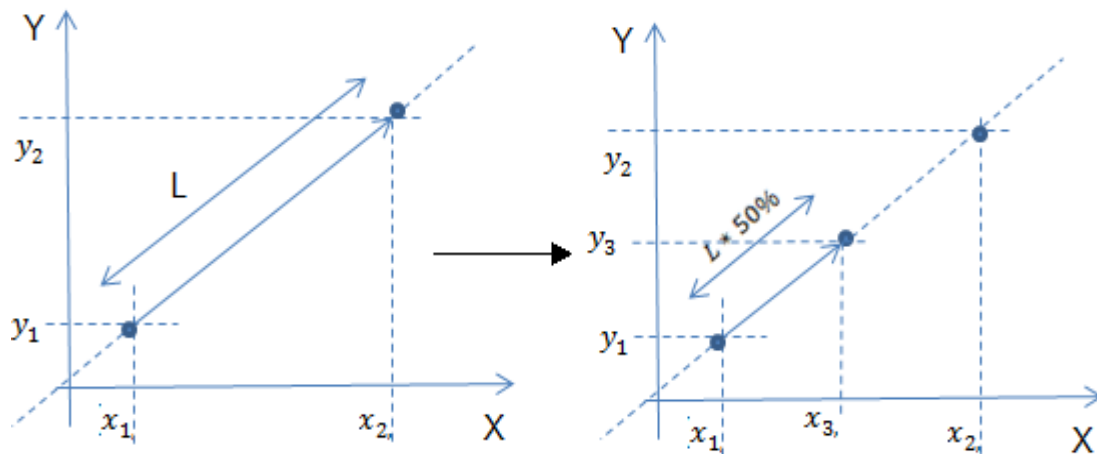


Figure 3.21: Length control

### 3.3.3 Nodes Placement

Since partially drawn link is deal with straight line, the graph has to be a kind of straight line drawing. Besides, the end of link with arrow head plays a role as direction guider but not closely touch the target node, nodes in graph must keep well distance to avoid misleading. One embarrassing example is in Figure 3.21, target nodes are getting too close, partial links lost duty of direction guiding. If irrelevant nodes be drawn closely and connected nodes are place too far away could cause viewer confusion also.

The initial graph layout is important for this simple partially drawn link, new graph drawing algorithm will not develop here, but use force-directed drawing which was introduced in section 3.2. Because nodes in force-directed graph keep well distance, pairwise nodes be drawn near each other and it's static equilibrium can reduce computation time than moving nodes. Original graph in equilibrium state be drawn and then shorten the links. To make full effect of this technique and accommodate different tasks, a changeable parameter or slider bar is used to control length of all links, viewer can make comparison between original graph and graph with shortened links. Figure 3.23 make such comparison between before and after links shortened, edge crossings can be reduced or no edge crossing.

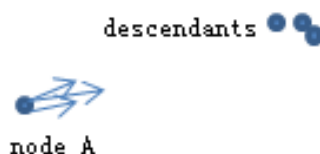


Figure 3.22: failure example

Force directed algorithm requires high running time, but it's still a good method for graph drawing when there're more and more nodes. To reduce computation time and make our partially drawn links more efficient, we place nodes in a

simpler way when there are not too many nodes.

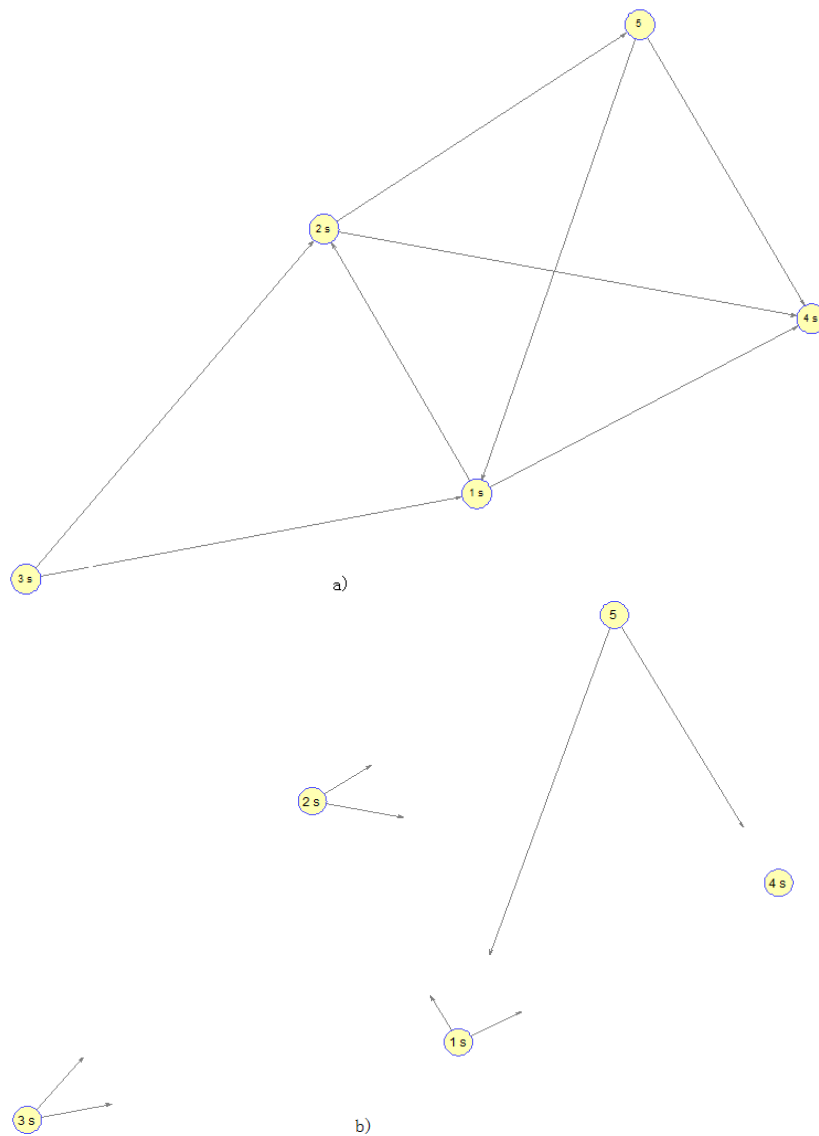


Figure 3.23: a) Force-directed layout  
b) partially drawn links

Inspired by circular layout, given an analog circle, according to the number of nodes, use angle control to place nodes evenly on the circle. Graph symmetry can be maximized and nodes overlap can be prevented also. It's efficient especially not too many nodes but lots of edges.

Assume we have  $m$  nodes, then circle be divided to  $m$  parts, each one with angle  $\alpha = 2\pi/m$ , then the coordinate point of node decided by  $\sin \alpha$  and  $\cos \alpha$ .



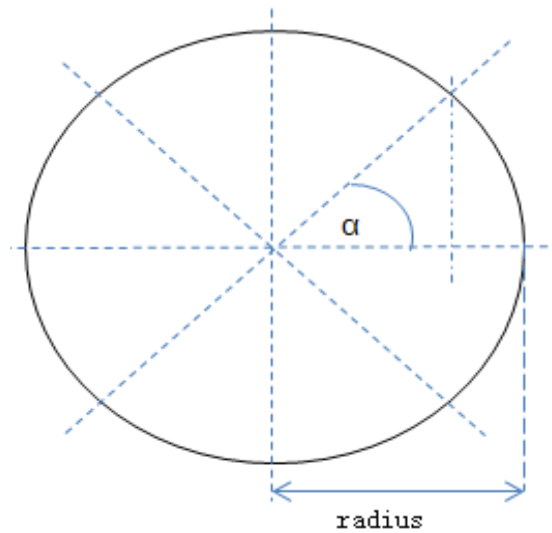


Figure 3.24: circular placement

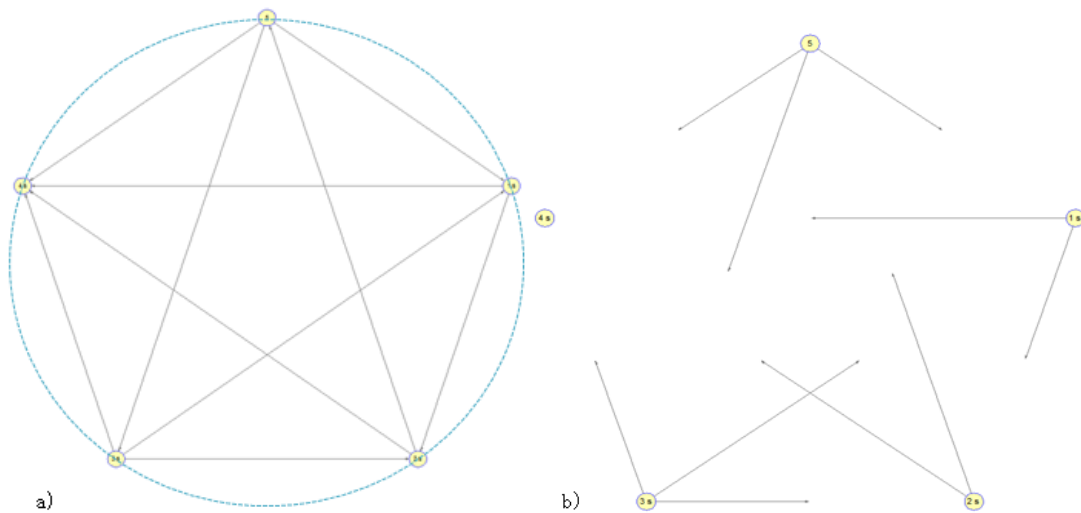


Figure 3.25: a) before shorten links

b) after links shortened

Pseudo-code of core body:

*Initial straight-line drawing, position of each node:  $[x_i, y_i]$*

*For  $i \rightarrow 1$  to  $n$*

*For  $j \rightarrow 1$  to  $n$*

*Intermnodepos =  $[(y_k - b)/k, y_k = y_j + (y_j - y_i)/a]$*

*Or =  $[x_k = x_i + \frac{(x_j - y_i)}{a}, k * x_k + b]$*

*// Arrow head position calculation*

*end*

*end*

The running time of the algorithm depends on the number of links, that is,  $O(|E|)$ . The running time of the whole program is  $\theta(|E|)$  plus initial straight line drawing time.

### 3.3.4 Partially Drawn Links in User-defined Region

All graphs presented above are all static pictures, and static graphs provide limit information to viewer. The subway maps in Figure 3.1 and 3.2 for example, besides graphs base on drawing algorithms, other techniques like color coding and remark is used to enable viewer to find interesting information. But it`s not enough. When density of a graph is large, or consider realistic problem like viewer with near-sightedness, details of graph may not be observed clearly. The components of node-link graph are nothing else but nodes and links, easy to cause illusion chaos even with not too much nodes and edges.

‘A good visualization is not just a static picture that we can walk through and inspect like a museum full of statues’<sup>[17]</sup>. Various operations is possible with the help of computer interactive interface, enable user to get inside the graph and look for something important or interesting. Schneidermann<sup>[28]</sup> summarize the user`s information searching behavior as: ‘Overview first, zoom and filter, then details on demand’. He presented seven tasks for graph visualization, which may fulfill all what viewer needs:

- Overview: Gain an overview of the entire collection.
- Zoom: Zoom in on items of interest.
- Filter: Zoom in on items of interest.
- Details-on-demand: Select an item or group and get details when needed.
- Relate: View relationships among items.
- History: Keep a history of actions to support undo, replay, and progressive refinement.
- Extract: Allow extraction of sub-collections and of the query parameters.

Of course, viewer don`t need so many functions usually. We offer two information filtering operations in the software interface.

1. Zooming. Zooming is a common operation, zoom in to see details of graph and zoom out have overview. In figure 3.23 for example, node ID or even arrow may not clear under some circumstances. Our interface provides zoom in

and out buttons:  .

2. It`s not enough just for zooming in and out. The second operation is user-defined region. User may just interest in one or several nodes` information. By the use of mouse, click to select a rectangle region which includes nodes user-demanded to process partially drawn their links. The example we used in Figure 3.11, sometimes we just know want to know who a particular person ever contacts, select node represent this person, only out-links of this node will be shortened. In other words, only these particular user-demanded information be filtered out.

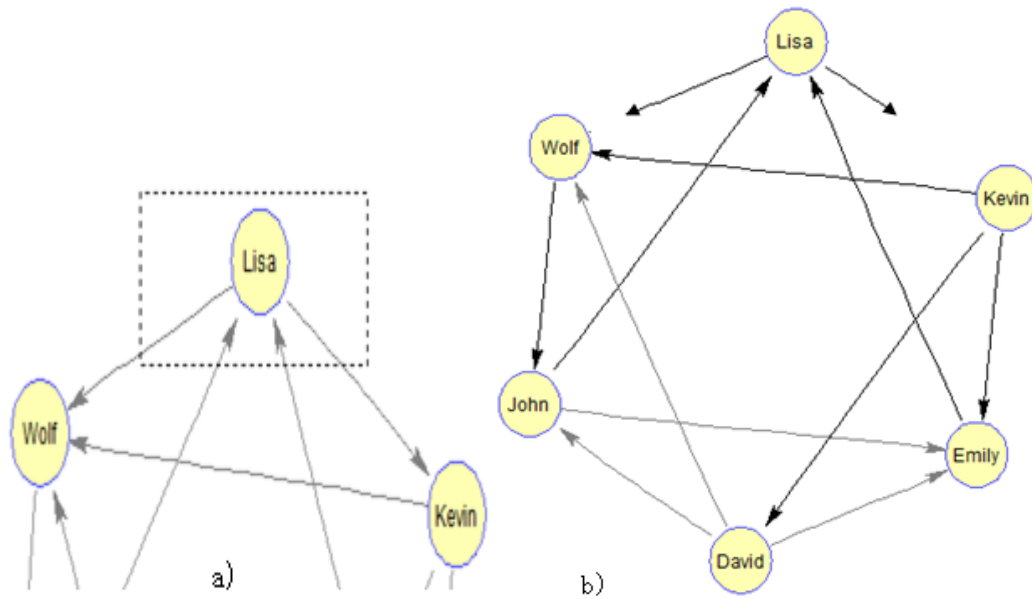
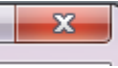


Figure 3.26: a) region selected

b) partially drawn links for region

Besides, our interactive interface displaces both original graph and partially drawn links graph at the same time, user can make comparison of both graphs. At some software, previous graph would close automatically when new graph displace, this method keeps interface clear and neat. Both method has it's own

advantage. We choose to let user click 'close'  icon to close down graphs don't needed.

Our user interface provides a more ideal method is to use a slider bar to control the length of links, let links change proportional, it's much more match viewer's perception. It's convenient for user control and observe the length, easy-operate just by drag the slider bar up and down.

### 3.3.5 Discussion and Further work

The realization of partially drawn links still has many restrictions, that means there is substantial room for improvement:

1. Image detection technique can be introduced, allow user import straight-link graph which nodes in random places to process partially links drawing.
2. If edges close to or cross through a certain node, node can be moved to ideal new position automatically.
3. Curved and straight lines being used at the same time usually. We can remain the curve but shorten all straight lines, beauty and readability of graph would be enhanced.

4. Animation is another important technique could be embedded. For nodes which user selected, like an electric torch, keep switching on and off, links look like light shoot out, the position where light could reach is the length of link.

### 3.4 Hierarchical Clustering Algorithm

Unlike the above algorithms, the nodes processed by clustering algorithm do not have to be connected by edges. This algorithm is based on statistical analysis, grouping objects according to their similarity, and each group shares the same property or characteristics. Clustering is a kind of unsupervised process of grouping a more understandable presentation of input data<sup>[30][31]</sup>.

Hierarchical clustering algorithm has widely application in world wide web, social network, marketing analysis and so on. Statistical analysis usually results large amount of data, it's not intuitive to represent by language, forming or listing, especially inconvenient when we just need an overview. Hierarchical clustering results a tree viewed dendrogram, which cooperate with interpreted node-link graph.

Distance matrix being used to describes distance between objects, entities of this matrix is the distance between objects and the similarity between objects is the study on relationships between entities of distance matrix. Weighted adjacency matrix is used as distance matrix here.

There are two important components of clustering, one is clustering algorithm, and the other is linkage criteria.

#### 3.4.1 Agglomerative Clustering

Agglomerative Clustering works as merge two clusters into one base on linkage measure. It's initial condition is to view every object as a single cluster, merge to final one cluster after some iterations.

Given a set of  $N$  objects to be clustered, and an  $N*N$  distance matrix, the basic process of hierarchical clustering<sup>[28]</sup> is:

1. Assign each object to its own cluster, so we have  $N$  clusters initially, each cluster contains one object, and the distance between clusters equal the distance between objects.
2. Find the closest (has most similarity) pair of clusters and put them into a single cluster, now we have one less cluster.
3. Distances between new cluster and the remaining old clusters, then we have a new distance matrix now.
4. Repeat 2 and 3 until all clusters are put into a single cluster, and the size of the final cluster is  $N$ , which means it contains  $N$  objects.

The flow chat of clustering process is given below:

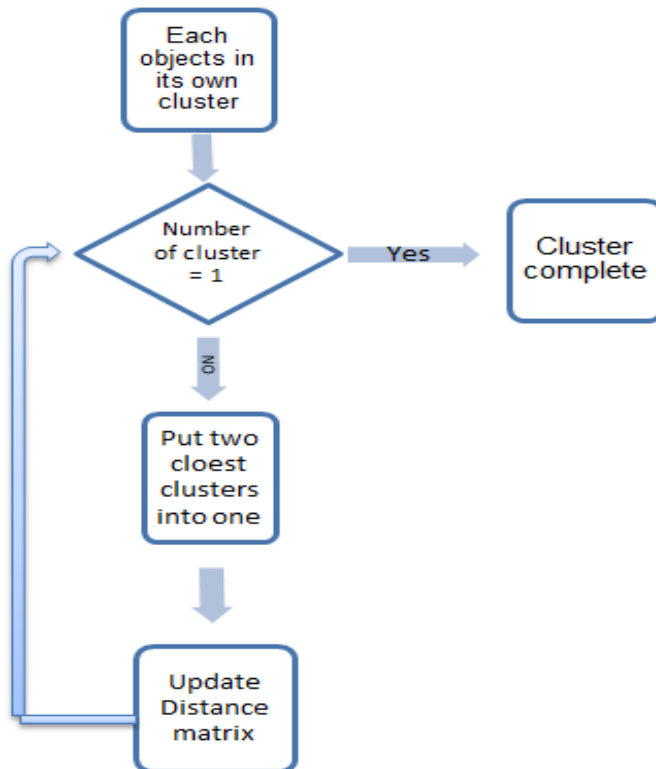


Figure 3.27: Flow chat of hierarchical clustering

Step 2 and 3 in hierarchical algorithm can be done in different ways, depend on which linkage measurement used for clustering.

### 3.4.2 Linkage Criteria

Distinct results be obtained by using different likage method. There`re three most commonly used criteria in agglomerative hierarchical clustering: minimum distance(Single linkage,), Maximum distance(Complete link)和 average distance(average linkage)<sup>[33]</sup>.

#### (a) Single linkage clustering

Distance between two clusters is the minimum distance from one point of its cluster to any point of the other cluster:

$$D = \min d(x_i, x_j)$$

Where  $d(x_i, x_j)$  is the entity of distance matrix. 'It's called single link because it says clusters are close if they have even a single pair of close points, a single link'<sup>[32]</sup>

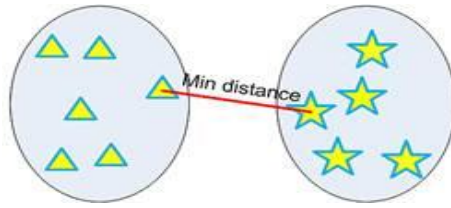


Figure 3.28: Single linkage

(b) Complete linkage

Distance between two clusters is the maximum distance from one point of its cluster to any point of the other cluster:

$$D = \max d(x_i, x_j)$$

It's similar to single linkage, just use maximum distance instead.

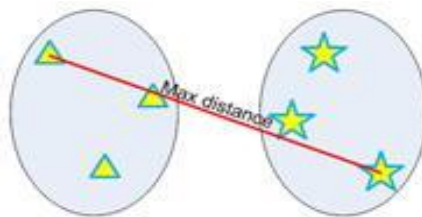


Figure 3.29: Complete linkage

(c) Average linkage

Distance between two clusters is computed as the average distance between nodes of one cluster and nodes of another cluster. 'Average linkage tends to join clusters with small variances, and it is slightly biased toward producing clusters with the same variance'<sup>[34][35]</sup> which was originated by Sokal and Michener(1958).

$$D = \frac{1}{2} \sum_{i \in C_K} \sum_{j \in C_L} d(x_i, x_j)$$

where  $C_K, C_L$  are two clusters, distance between members of these two clusters be calculated. 'Average linkage is viewed as a compromise between single and complete linkage, all nodes in cluster is considered but not just one node, has less influenced by extreme large or small number.'<sup>[36]</sup>

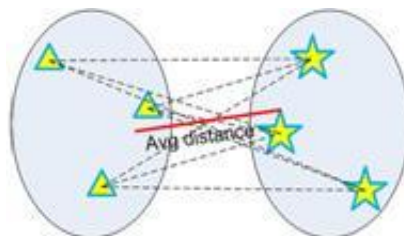
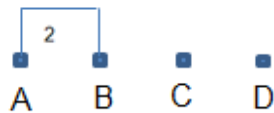


Figure 3.30: Average linkage

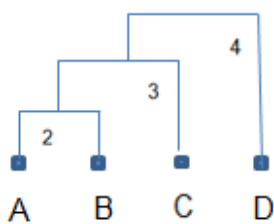
Let's have an example of how these three methods deal with a same weighted adjacency matrix input in different ways. Assume we have four sites: A, B, C, D, distance between these sites is given as:

	A	B	C	D
A	0	2	5	9
B	2	0	3	7
C	5	3	0	4
D	9	7	4	0

(a) Single linkage



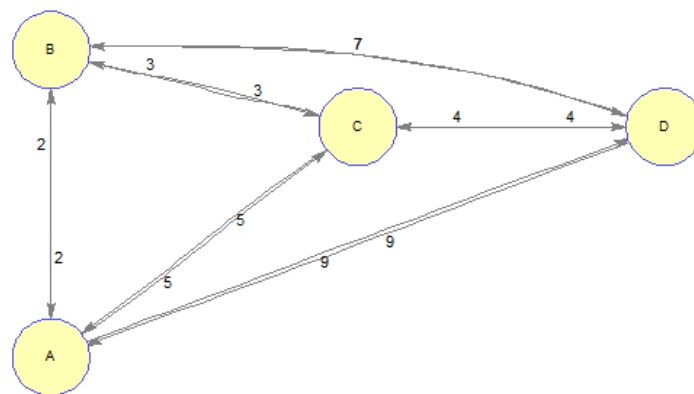
$$d((A,B), C) = \min\{d(A, C), d(B, C)\} \\ = \min\{5, 3\} = 3$$



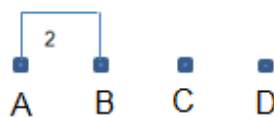
$$d((A,B), D) = \min\{d(A, D), d(B, D)\} \\ = \min\{9, 7\} = 7$$

$$d(C, D) = 4$$

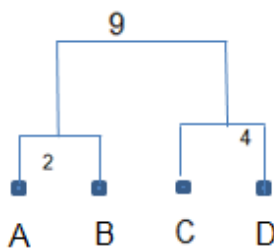
$$d((A,B, C), D) = \min\{d((A, B), D), d(C, D)\} \\ = \min\{7, 4\} = 4$$



(b) Complete linkage



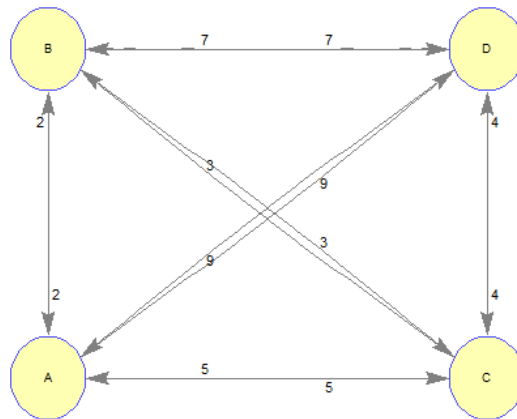
$$d((A, B), C) = \max\{d(A, C), d(B, C)\} \\ = \max\{5, 3\} = 5$$



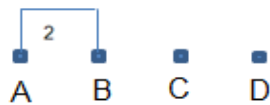
$$d((A, B), D) = \max\{d(A, D), d(B, D)\} \\ = \max\{9, 7\} = 9$$

$$d(C, D) = 4$$

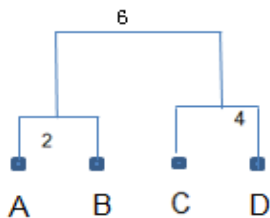
$$d((C, D), (A,B)) = \max\{d(C, (A, B)), d(D, (A, B))\} \\ = 9$$



(c) Average linkage



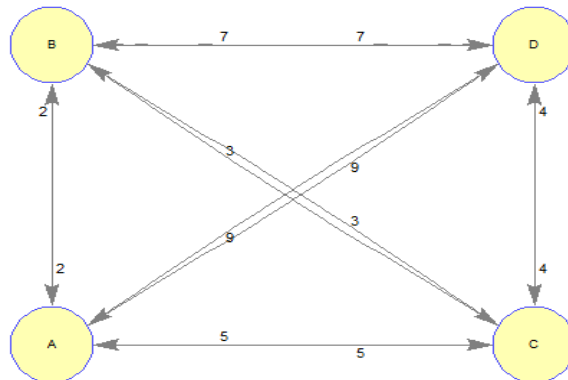
$$d((A,B), C) = \text{avg} \{d(A, C), d(B, C)\} = (5+3)/2 = 4$$



$$d((A,B), D) = \text{avg} \{d(A, D), d(B, D)\} = (9+7)/2 = 8$$

$$d(C, D) = 4$$

$$d((C, D), (A,B)) = \text{avg} \{d(C, (A, B)), d(D, (A, B))\} = (4+8)/2 = 6$$



### 3.4.3 Directed Graph

Directed graph is a must in this domain, because relationships between objects may not be mutual equivalent. Little work has been done on clustering deal with directed graph, most work to clustering target undirected situation.

The major challenge is that relationships in directed graph is different from undirected graph. This difference is obvious if compare the adjacency matrix representation. The symmetry property of undirected adjacency matrix let nodes in clusters share similar in and out links. The adjacency matrix of a directed graph is square but not symmetric. A simply way to via a matrix transformation:  $D_{undirected} = D_{directed} + D_{directed}^T$ , where  $D_{directed}^T$  is transpose of



$D_{directed}$ . The weight of the edge in the symmetrized graph will be the sum of the weights of the two directed edges<sup>[37][38][39]</sup>.

## 3.5 Strongly Connected Components of Directed Graph

### 3.5.1 Strongly Connected Components

Connective problem is rather simple and obvious in undirected graph, an undirected graph can be decomposed to several connected components<sup>[41]</sup>, see Figure 3.30. The depth first search which introduced in chapter 2 does this easily, each restart of search is a connected component.



Figure 3.31: Undirected graph

However, in directed graph, connectivity problem is more complicated, because the edge is 'one way street'. In directed graph, between two nodes A and B, if there exists a path from A to B, and a path from B to A, we call these two nodes is strongly connected. 'A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions'<sup>[40]</sup>, abbreviated as SCC. There could be multiple SCC in a directed graph, even a single node could be a SCC.

### 3.5.2 Searching Algorithm

After depth first search of a graph, it returns topological layers order, each SCC must be one sub-layer. Then we just need to find out the starting node of each layer, take out the SCC one by one.

Define the  $DFN[n]$  be the order label of nodes visited before depth first search, and  $low[n]$  be the smallest depth first search number of a node reachable by a back or cross edge from  $N$ 's subtree. The current node is a root of one SCC if  $DFN[n] = low[n]$ . Because if it's not a root of SCC, then it must belongs to another SCC and its root must be ancestor of current node, then a path to the ancestor and include the current node would exists, if so,  $low[n]$  has to be changed to a value smaller than  $DFN[n]$ .

Pseudo Code:

DFS(u)

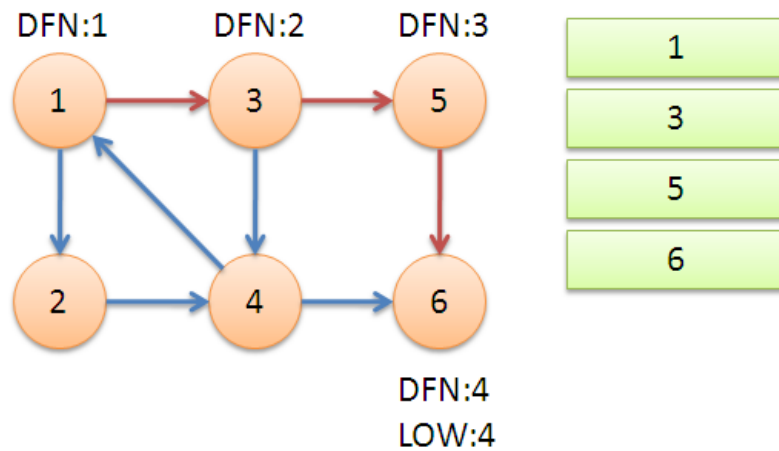
```

{
  DFN[n]=Low[n]=++Index          % set label for node n and initialize Low[n]
  Stack.push(n)                  % push node n into stack
  for each (n, v) in E           % for each edge
    if (v is not visited)       % if node v not been visited
      search(v)                  % continue
      Low[n] = min(Low[n], Low[v])
    else if (v in S)             % if node n still in stack
      Low[n] = min(Low[n], DFN[v])
  if (DFN[n] == Low[n])          % if node n is the root of SCC
    repeat
      v = S.pop                   %pop node v out of stack
      print v
    until (n== v)
}

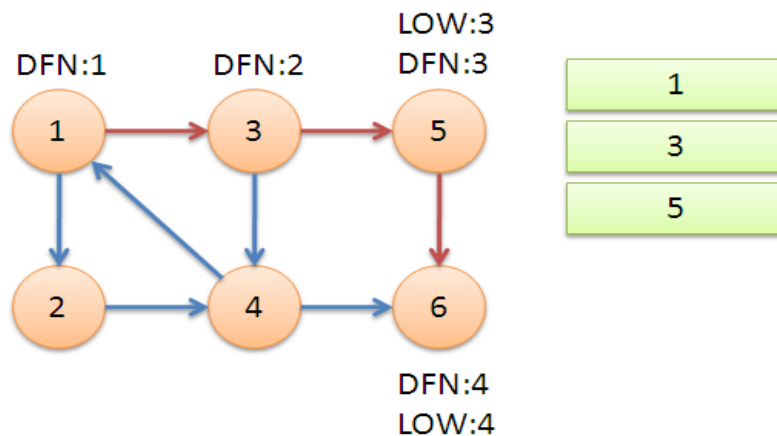
```

SCC searching is one of those algorithms which is difficult to be understood by language describe. The following is flowing steps representation:

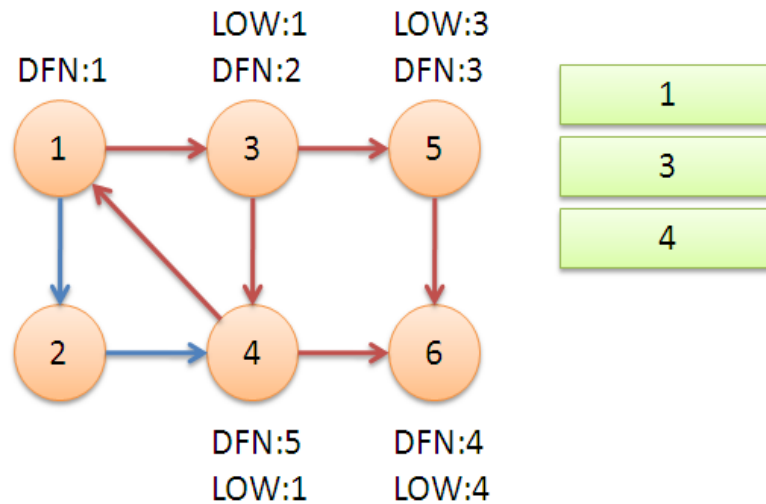
(1) Start DFS from node 1, push all traversed nodes into stack. When reach node 6, no out-edge from node 6 and  $DFN[6] = Low[6]$ , a SCC {6} is found.



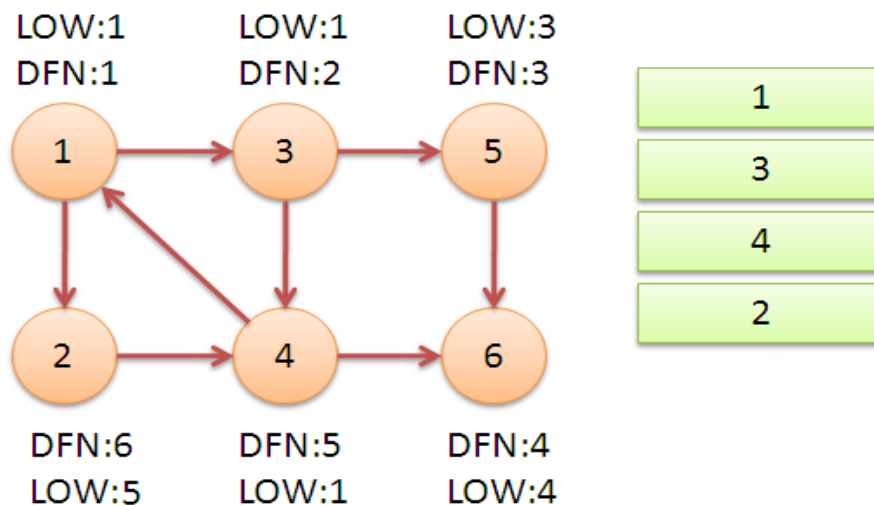
(2) Way back to node 5, and  $DFN[5] = Low[5]$ , push node 5 out of stack and it's a SCC {5}.



(3) Way back to node 3, it has an unsearched edge, reach node 4 and push it into stack. There're two out-edges from node 4, one is direct to node 1 which already in the stack, so  $Low[4] = 1$ . No more visit to node 6 because it's already out of stack. Way back to node 3.  $Low[3] = Low[4] = 1$ , because edge between node 3 and 4 is branch of the sub-tree.



(4) Way back to node 1, visit node 2 along the only unsearched out-edge. Because node 4 still in stack, so  $Low[2] = DFN[4] = 5$ . Way back to node 1, found that  $DFN[1] = Low[1]$  already, take all nodes in the stack out and form a SCC  $\{1,3,4,2\}$ .



All nodes been visited once, in and out of stack once, so the time complexity is  $O(n+e)$ , where  $n$  is the number of nodes and  $e$  is the number of edges.

### 3.5.3 Graph Layout of SCC

SCC has realistic application in transportation routes arrangement, social network user data analysis and so on. In graph visualization, could help to reduce the size of graph, see Figure 3.31, node 1,2,3 and 4 is one strongly connected component, then in some task they could be drawn into a single

node, the final graph just contains three nodes and three edges.

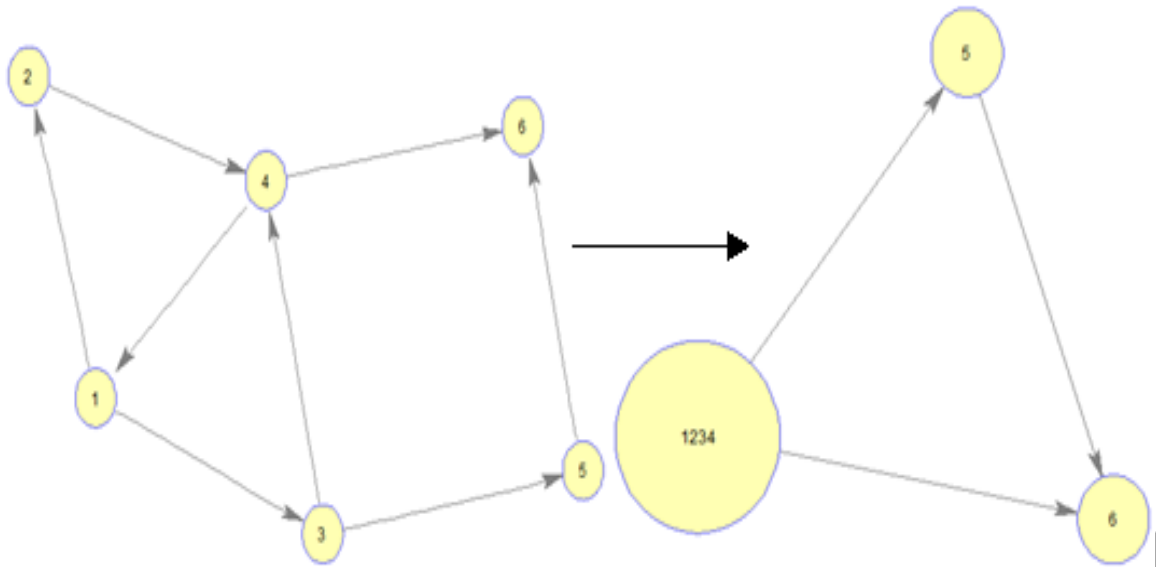


Figure 3.32: Graph size reduce

Simply color coding is introduced here to draw graph to distinguish SCCs of a graph. Apply HSV colormap which ranges from blue to red, and passes through the colors cyan, yellow, and orange. Assign different color to each SCC and the color is depends on the number of SCC. Result the Figure 3.32.

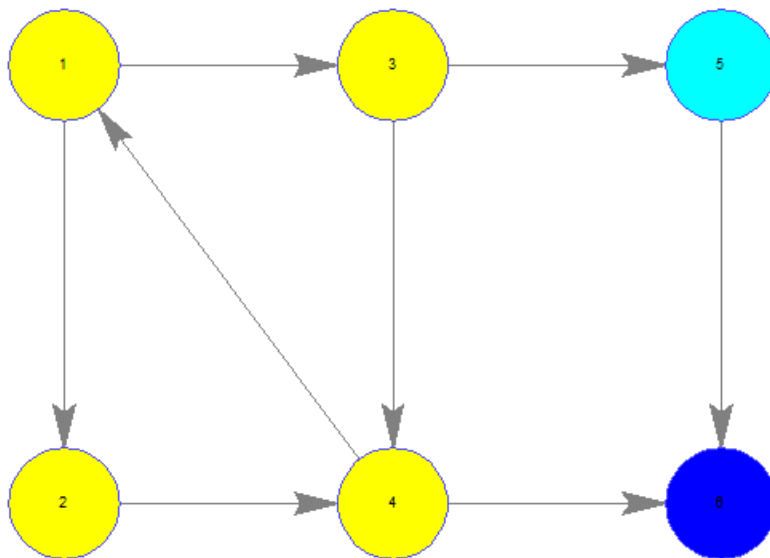


Figure 3.33: Color coding layout

### 3.6 Shortest Path between Two Nodes

We already introduced Floyd-Warshall algorithm to find all-pairs shortest path. But in application, it's difficult to draw graph base on the algorithm resulted matrix. Because the shortest path between two nodes is not necessary the single edge between two nodes. Besides, in daily life, what we usually want to figure out is just the shortest path between two nodes.

Assume we have  $n$  objects, weight of edge from node  $i$  to node  $j$  is  $W_{ij}$ . For any node  $k$ , the shortest path from node  $i$  to node  $j$  may pass or not pass node  $k$ . If  $d_{ik}$  is the shortest distance from node  $i$  to node  $k$  and  $d_{kj}$  is the shortest distance from node  $k$  to node  $j$ , compare  $d_{ij}$  and  $d_{ik} + d_{kj}$ , if  $d_{ij} > d_{ik} + d_{kj}$ , let  $d_{ij} = d_{ik} + d_{kj}$  to keep  $d_{ij}$  always be the shortest distance from node  $i$  to node  $j$ . Repeat until all node  $k$  are searched, the final  $d_{ij}$  is what we want. The detailed process is:

Step 1 Input weight adjacency matrix  $W$  and initialize  $d_{ij} = 0$ .

For all  $i, j$  we have  $d_{ij} = W_{ij}$ ,  $k=1$ .

Step 2 Update  $d_{ij}$ . For all  $i, j$ , if  $d_{ij} > d_{ik} + d_{kj}$ , let  $d_{ij} = d_{ik} + d_{kj}$ .

Step 3 Stop when  $k=n$ , otherwise, go to step 2.

Pseudo Code:

```
// dist(i,j) is the shortest distance from node i to node j
For i←1 to n do
  For j←1 to n do
    dist(i,j) = weight(i,j)

For k←1 to n do // intermedia node is the primary condition
  For i←1 to n do
    For j←1 to n do
      if (dist(i,k) + dist(k,j) < dist(i,j)) then
        dist(i,j) = dist(i,k) + dist(k,j)
```

Figure 3.34: Shortest path between two nodes<sup>[11]</sup>

Figure 3.33 is the core body of Floyd-Warshall algorithm, the advantage of this algorithm is easy to understand and implement. Finding shortest path has practical meaning in traffic route arrangement, transportation route planning and so on. For example, there're five cities, assign distance or travel expense from one city to another as weight value, shortest or cheapest way from city C to city F can use graph to visualize this information as:

	A	B	C	D	E	F
A	0	4	0	1	0	7
B	2	0	3	7	2	1
C	5	3	0	4	0	10
D	9	7	4	0	5	0
E	8	6	4	6	0	1
F	3	6	2	8	0	0

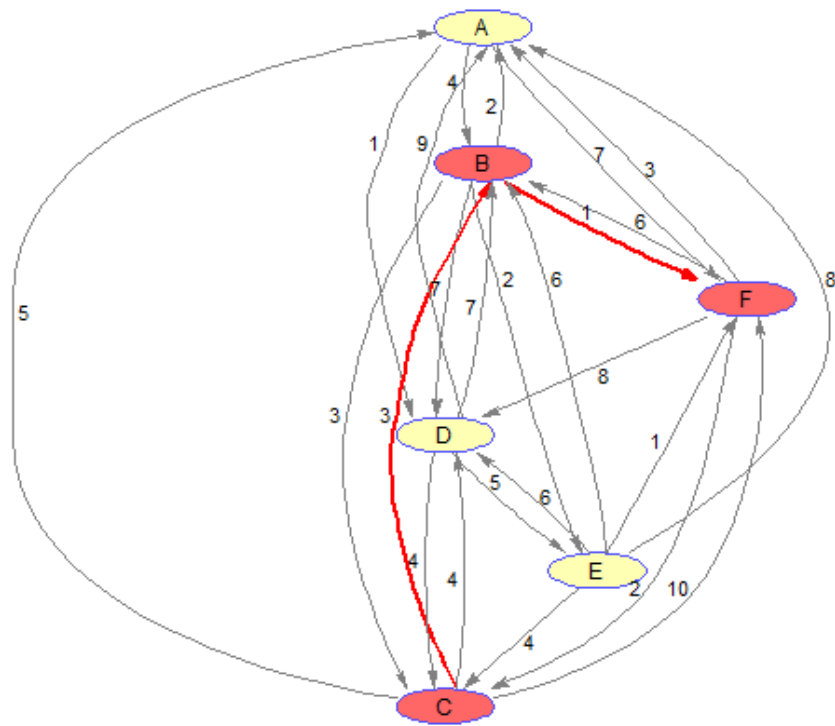


Figure 3.35: Shortest Path from City C to City F

## Chapter 4 Matlab GUI Instructions

GUI and algorithms implementation are all programmed by the software MATLAB. The GUI is shown up by clicking a .exe executable file, here's the initial interface:

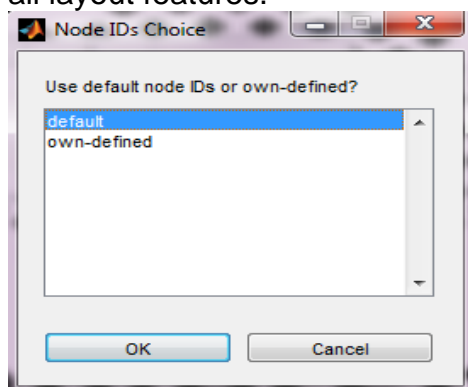


There are six main menus on the top:

1. Data Import;
2. Layout;
3. Partially Drawn Links;
4. Find Strongly Connected Components;
5. Hierarchical Clustering;
6. Shortest Path between 2 Points.

(a) Data Import

Click to import data from hard disk, a pop-up dialog box enables the user to use default IDs or input user-defined node IDs. This preferred IDs will be applied to all layout features.



A grid box will display the input adjacency matrix with the node IDs. To reduce the edges, nodes which have self-connections are not present by edge but add a 's' in the node ID instead. For example:

	1 s	2 s	3 s	4 s	5
1 s	1	1	0	1	0
2 s	0	1	0	1	1
3 s	1	1	1	1	0
4 s	0	0	0	1	0
5	1	0	1	1	0

### (b) Layout

Four types of graph can be layout according to input data:

Show Graph(default edge type)
Straight Line
Force-Directed(Undirected)
Equilibrium Layout(directed)

### (c) Partially draw links

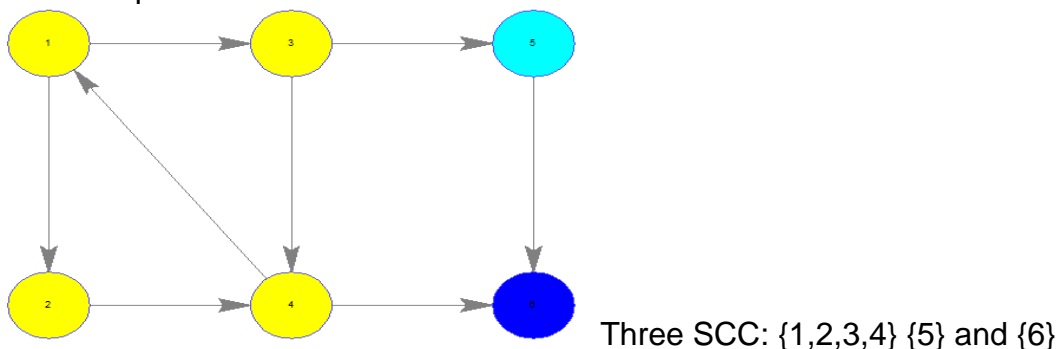
Two kinds of initial graph provided: Equilibrium state and place node on circle evenly.

Two kinds of links length control methods also coexist:

1. Enter user-defined 'shorten parameter' then original graph and graph after links shortened display. Left-click your mouse, hold and drag a rectangle area, partially drawn links will process the nodes inside the selected area.
2. An easy-operated slider bar used to control links length, original graph and graph after links shortened are all provided.

### (d) Find Strongly Connected Components

Graph's strongly connected components will layout with color coding.  
For example:



### (e) Hierarchical Clustering

Clustering input data using agglomerative algorithm, three methods provide:

- 1, single linkage
- 2, complete
- 3, average



Dendrogram along with node-link diagram cooperate to layout hierarchical clustering of data.

(f) Shortest Path between 2 Points

Click to select, select the source and destination nodes from pop-up nodes list, shortest path be highlighted in graph.

## Chapter 5 Conclusion, Discussion and Future Work

### 5.1 Performance

Performance test is running at laptop: Intel Core i3-2350M(2.3GHz), Windows 7 64bits, 1GB memory. Graph drawing not just affected by algorithms` calculation time, but also the property of drawing frame: `figure` file.

The drawing frame can`t be infinite large, one major reason is computer processing ability and the other is for a good layout. Our drawing frame area is set to  $X \times Y \approx 1000 \times 1000$  (unit area). Each node occupies  $20 \times 20 = 40$  unit area, that means  $1000 \times 1000 / 20 \times 20 = 2,500$  nodes could be placed theoretically. Of course, edges and arrow should not be ignored in graph drawing. Assume each node had an in and an out edge, each unit needs about  $40 \times 40$  unit area, otherwise, some tricky problems may occur: node shape changed, node not at predefined position and so on.

In fact, after testing we find out that edge is an extremely important factor when layout a graph. If a graph with more nodes and less edges, can deal with approximate 130 nodes and 300 edges. On the contrary, just approximately 50 nodes and 900 edges could be handled.

For force-directed undirected graph, dot to represent objects instead of circle node, space occupation can be reduced and layout speed can be increased. It could layout graph with up to about 130 nodes and 10,000 edges.

### 5.2 Discussion and further work

Matlab is a powerful software because of its comprehensive functions, but rather a little bit weak in running time and self-update. Java is comparatively complex in programming but faster running time. Matlab used more at base studying while JAVA implement tasks much better. JAVA version of partially drawn links may lead to better performance.

During performance testing, two restrictions of partially drawn links found: one is when two shortened links` arrow heads intersect at one point, the `figure` frame would misplaces them automatically sometimes. The other is the way we shorten the links. Intermedia node coordination point calculation may needs up to many digits after the decimal point, error may occurs, depends on the parameter we chose. The same problem as force-directed, constant C is experimental based also. Better algorithm can be presented to improve the partially drawn links.

Graph theories actually algorithms to traverse a graph, I start graph drawing study with graph theory but did not apply it much in graph drawing technique except SCC and shortest path. We study how input data matrix affect graph layout, combine research of data and graph drawing by learning graph theory.

Combination of graph theory and various node position and drawing algorithms can help us to draw more appealing graph. For example, DFS we learned before, results nodes` topological order, cooperate with SCC and even hierarchical clustering, edge crossing can be minimizing and nodes positioned in more human sense reasonable way<sup>[11][40]</sup>. But mentioned before, none of drawing method is `perfect`, it`s necessary for visualizator try to learn and implement different methods.

Besides, hierarchical clustering can be combined with many nodes position algorithms and graph theory also, like force-directed, circular drawing. Drawing each cluster in force-directed algorithm can improve graph meaning. Actually, k-means, flow cut and other techniques already used in hierarchical clustering, improving layout in different meaningful graphs<sup>[42]</sup>.

We select adjacency matrix form as input data, actually, it`s better if the graph visualization interface can support more input data form, like adjacency list, sparse matrix. One reason is enable user has more choices, and different input data can directly affect algorithm performance.

There is still substantial room for improvement and develop in interactive features. More user-defined operation to nodes and edges could be developed. For example, enable user to drag nodes to anywhere they want by mouse, but edges go together along the nodes. That is, nodes position changed but connection remains. User can move edges around the frame, change their shape and size at will without change edges` two ends.

Graph visualization is devoted to reveal data relationship and meaning, let viewers observe what they interest in and find out unknown. How to build a easily operational interactive interface, combine more features into one block, especially enhance the automatically generate ability are important topics for visualizator.

## Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 30. März 2012 \_\_\_\_\_

## Acknowledgements

I want to thank Dr. rer. nat. Michael Burch, my supervisor, for his advice, patience and understand, provide me with materials, valuable ideas and his precious time. Guide me from blank to indoor, show me a whole new and colorful scientific field. A little bit shame that I did not do well enough. For giving me strong support and tolerance, Prof. Dr. Daniel Weiskopf and Ms. Ritzmann. For giving our INFOTECH students generous support and helpful suggestions these years, Dr.-Ing. Manfred Wizgall, my course director.

I shall extend my thanks to all my friends, take care my daily life during these days, especially when suffered from physical injury and laptop damage.

Finally, I owe a lot to my parents, who have given me life and love, consistent support and encouragement.

## Bibliography

- [1] Weiwei Cui: A Survey on Graph Visualization. Hong Kong University of Science and Technology, 2005.
- [2] A Math Forum Project, Drexel University, 1994-2012.
- [3] D. E. Knuth: Computer-drawn Flowcharts. Communications of the ACM, 1963.
- [4] THOMAS M. J. FRUCHTERMAN AND EDWARD M. REINGOLD: Graph Drawing by Force-directed Placement. SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 21(1 1), 1129-1164 (NOVEMBER 1991).
- [5] J. A. Bondy and U. S. R. Murty: Graph Theory with Application. Department of Combinatorics and Optimization, University of Waterloo, 1976.
- [6] Colin Ware, Helen Purchase, Linda Colpoys and Matthew McGill: Cognitive Measurements of Graph Aesthetics, University of New Hampshire, 2002.
- [7] Chaomei Chen: Information Visualization (beyond the horizon), Second Edition, 2006.
- [8] Sung-Chul Han, Franz Franchetti, and Markus Puschl: Program Generation for the All-Pairs Shortest Path Problem. Electrical and Computer Engineering, Carnegie Mellon University Pittsburgh, PA 15213, 2006.
- [9] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest Introduction to Algorithms, 2009.
- [10] Cliff Stein: Optimization II. Columbia University, 2006.
- [11] Cormen, Leiserson, Rivest, and Stein. Introduction to Algorithms (2<sup>nd</sup> Edition). MIT Press, 2001.
- [12] Stephen R. Schmitt: Mathematics and Software Tutorial, 2010.
- [13] Michelle Girvan and M.E.J. Newman: Community structure in social and biological networks, 2001.
- [14] Chris Bennett, Jody Ryall, Leo Spalteholz and Amy Gooch: The Aesthetics of Graph Visualization. Computational Aesthetics in Graphics, Visualization, and Imaging, 2007.
- [15] Helen C. Purchase: Metrics for Graph Drawing Aesthetics, 2002.

- [16] Takao Nishizeki, Md. Saidur Rahman: Planar Graph Drawing. Lecture Notes Series on Computing, 2004.
- [17] Colin Ware: Information Visualization Perception for Design, 2004.
- [18] Eades. P: A heuristic for graph drawing. Congressus Numerantium 42, 1984
- [19] Richard Fitzpatrick: Classical Mechanics. The University of Texas at Austin, 2006.
- [20] Jennifer Golbeck and Paul Mutton: Spring-Embedded Graphs for Semantic Visualization, 2005.
- [21] John Howse, Peter Rodgers and Gem Stapleton: Automated Diagram Drawing, VL/HCC 2009.
- [22] Michael Kaufmann and Dorothea Wagner: Drawing Graphs-Methods and Models, 2001.
- [23] Stephen G. Kobourov. Force-Directed Drawing Algorithm. University of Arizona, 2004.
- [24] Charles D. Hansen, Chris R. Johnson: The Visualization Handbook, 2005.
- [25] Yidong Feng, Guoping Wang and Shihai Dong. Information Visualization. Journal of Engineering Graph, 2001.
- [26] Farin, Gerald: Curves and surfaces for computer-aided geometric design, 1996.
- [27] Bridgeman, S., Fanto, J, Garg, A. Tamassia, R., and Vismara, L. (1997): InteractiveGiotto: An algorithm for interactive orthogonal graph drawing, 1997.
- [28] Ben Shneiderman: The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization. University of Maryland, 1996.
- [29] S. C. Johnson: Hierarchical Clustering Schemes, 1997.
- [30] Miklós Erdélyi, János Abonyi: Node Similarity-based Graph Clustering and Visualization, 2006.
- [31] Olaf Sporns: Graph Theory Methods For The Analysis Of Neural Connectivity Patterns, 2002.
- [32] Ryan Tibshirani: Distances between Clustering, Hierarchical Clustering, 2009.

- [33] Kardi Teknomo: Linkages between Objects, 2009.
- [34] Pankaj K. Agarwal: Algorithms in Computational Biology, 2003.
- [35] James L. Schmidhammer: Applied Multivariate Methods, 2011.
- [36] Daniel P. Berrar, Werner Dubitzky and Martin Granzow: A Practical Approach to Microarray Data Analysis, 2003.
- [37] Venu Satuluri, Srinivasan Parthasarathy: Symmetrizations for Clustering Directed Graphs, 2011.
- [38] Jie Chen and Yousef Saad: Dense Subgraph Extraction with Application to Community Detection. IEEE Transactions on Knowledge and Data Engineering (TKDE), 24(7):1216--1230, 2012.
- [39] David Gleich: Hierarchical Directed Spectral Graph Partitioning. Stanford University, 2005.
- [40] Tarjan, R.E: Depth first search and linear graph algorithms, 1972.
- [41] Christos Papadimitriou: Efficient Algorithms and Intractable Problems, 2012.
- [42] Adam Smith: Algorithm Design and Analysis, 2008.