IMPROVED NEIGHBOURHOOD SEARCH-BASED METHODS FOR GRAPH LAYOUT

By

Fadi Dib

June 2018

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT
IN THE SUBJECT OF COMPUTER SCIENCE
FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

Abstract

Graph drawing, or the automatic layout of graphs, is a challenging problem. There are several search-based methods for graph drawing that are based on optimising a fitness function which is formed from a weighted sum of multiple criteria. This thesis proposes a new neighbourhood search-based method that uses a tabu search coupled with path relinking in order to optimise such fitness functions for general graph layouts with undirected straight lines. None of these methods have been previously used in general multi-criteria graph drawing. Tabu search uses a memory list to speed up searching by avoiding previously tested solutions, while the path relinking method generates new solutions by exploring paths that connect high quality solutions. We use path relinking periodically within the tabu search procedure to speed up the identification of good solutions.

We have evaluated our new method against the commonly used neighbourhood search optimisation techniques: hill climbing and simulated annealing. Our evaluation examines the quality of the graph layout (fitness function's value) and the speed of the layout in terms of the number of the evaluated solutions required to draw a graph. We also examine the relative scalability of our method. Our experimental results were applied to both random graphs and a real-world dataset. We show that our method outperforms both hill climbing and simulated annealing by producing a better layout in a lower number of evaluated solutions. In addition, we demonstrate that our method has greater scalability as it can lay out larger graphs than the state-of-the-art neighbourhood search-based methods. Finally, we show that similar results can be produced in a real world setting by testing our method against a standard public graph dataset.

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my supervisor, Dr. Peter Rodgers, for his encouragement and guidance over the duration of my PhD. His constructive feedback and advice were the main reasons for the successful completion of this work. I have learned invaluable research skills under his supervision. I also thank him for giving me the opportunity to participate in a Dagstuhl seminar which was a memorable experience that I will always treasure.

My gratitude also goes out to Prof. Sally Fincher, Prof. Alex Freitas, and Dr. David Barnes for the valuable feedback and advice they provided during all the review sessions that improved the quality of my work. I would like to extend my gratitude to all the administrative staff in the School of Computing at the University of Kent for all the facilities they offer to students.

Special thanks go to Dr. Mahamed Omran for his recommendations in the early stages of this work. Those recommendations had opened the door for implementing the key algorithm discussed in this work.

Finally, and most importantly, I am really grateful to my parents, my sister, my brother, my wife, and my lovely daughter, who had supported me and inspired me in all my endeavours. Without their emotional support, prayers and love, I could have never made it this far. Many thanks to all of them for always being there whenever I needed them the most. May GOD bless you.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1 Introduction

This thesis addresses the problem of automated graph drawing for general graphs with undirected straight lines based on weighted sum multi-criteria optimisation. Graph drawing is the process of transforming a graph into a visual representation that is called a graph layout (di Battista et al. 1999). The graph layout depends on different aesthetic measures that could give a better understanding of graphs. Such aesthetic include edge crossings, edge length, node-to-node and node-to-edge occlusions, graph symmetry, angular resolution, and others (di Battista et al. 1999; Davidson & Harel 1996; Stott et al. 2011; Eades 1984). These aesthetics are measured and combined to form a multi-criteria weighted sum fitness function that measures the quality of a graph and is then optimised by search-based methods (optimisation methods).

Search-based methods can be placed into two categories according to the number of solutions examined at the same time: neighbourhood search-based methods and population-based methods. While neighbourhood search methods work on a single solution at a time, population-based methods evolve a set of points in the search space (Blum & Roli 2003). These methods can produce good solutions, but they have great potential for improvement. For example, in the case of neighbourhood search methods, simulated annealing adds an element of non-determinism in order to escape from local optima in the search space. This slows down the performance of the algorithm since this stochastic behaviour means that a large number of iterations can be required to reach a good solution (Davidson & Harel 1996). Hill climbing is generally faster in reaching a final layout, but the final result is not always the best as it is more likely to get trapped in a local optima (Talbi & Muntean 1993). Population-based methods, such as genetic algorithms, typically have an even slower rate of convergence compared to simulated annealing and hill climbing as they involve a wider search of the problem space. In addition, they often require large memory to maintain the population and can require additional algorithms to spread the solutions (Nam & Park 2000).

Our work in this field aims to address the problem of multi-criteria graph layout with a weighted sum fitness function from the perspective of neighbourhood search-based methods. To achieve this we have explored improved techniques that overcome the disadvantages of the current state of the art in neighbourhood search techniques. We propose a new neighbourhood search-based method that uses tabu search coupled with path relinking for drawing general graph layouts with undirected straight lines. None of these methods have been previously used in general multi-criteria graph drawing. Our method has two main features that distinguish it from other techniques: the use of a memory list to speed up searching by avoiding previously tested solutions; and the generation of new solutions by exploring paths that connect high quality solutions. We show that our method outperforms the current state of the art in neighbourhood search methods when being applied on randomly generated datasets and real world datasets.

1.1 Motivation and Objectives

Automatic graph layout is a topic in computer science that can be used in different applications from different fields. For example, Cerebral (Barsky et al. 2008) is a system that uses a biologically guided graph layout and incorporates experimental data directly into the graph display. Systems biology is a model for biological experimentation affected by the behaviour of thousands of biological entities that influence each other. These interactions are modelled as a graph, where the nodes represent biomolecules such as proteins and genes, and the edges represent interactions between them. Cerebral is used to lay out the graph model to interpret the results of experiments that will help biologists further refine the model. Our visualisation tool, described in Chapter 3, can be used as a replacement graph drawing back-end in tools such as Cerebral.

Many graph layout algorithms in the literature used neighbourhood search-based methods for drawing multi-criteria graph layouts, such as simulated annealing (Davidson & Harel 1996; Brank 2004; Lin et al. 2011; Gibson et al. 2013) and hill climbing (Stott et al. 2011; Talbi & Muntean 1993; Rosete-Suárez et al. 1999). Tabu search and path relinking were used in the field of graph drawing as well, but for single-criterion graph layouts (Marti 1998; Laguna & Marti 1999). On the other hand, population-based methods have also been used in drawing multi-criteria graph layouts with genetic algorithms (Kosak et al. 1991; Kosak et al. 1994; Branke et al. 1996; Eloranta & Mäkinen 2001).

Another popular type of automatic layout is the class of force-directed approaches. These differ considerably from search-based methods. Here, interactions between nodes are applied, such as the attraction of connected nodes and the repulsion of disconnected nodes, where the method attempts to find an equilibrium layout (Noack 2007; Gansner et al. 2013; Jacomy et al. 2014; Ortmann et al. 2016). In addition, systems such as Pajek draw large networks using spring embedders and eigenvectors (Batagelj & Mrvar 1998). However, aesthetics can only be indirectly coded in force-directed approaches, whereas search-based methods have the advantage of allowing tuneable combinations of directly coded metrics to meet user preferences.

We are not interested in finding the best possible layout, but enhancing the search mechanism is our main motivation. We want to improve the efficiency and effectiveness of neighbourhood search methods for drawing general graph layouts with undirected straight lines based on a weighted sum multi-criteria optimisation. The objective of our work is concerned with developing a new graph drawing search method based on tabu search and path relinking. These methods have not been used before to lay out general graphs with multi-criteria optimisation.

Tabu search is a neighbourhood search-based technique which proceeds on the assumption that selecting an inferior solution is not beneficial unless it is necessary such as escaping from a local optimum (Lim & Chee 1991). Tabu search keeps information on the itinerary through the last solutions visited. The role of this is to restrict the choice of some subsets in the neighbourhood by forbidding moves to some neighbour solutions that have already been visited (Hertz, et al. 1995).

Path relinking integrates intensification and diversification strategies (Glover et al. 2000). This approach generates new solutions by exploring paths that connect high quality solutions (elite solutions from the reference set) by starting from one of these solutions, called the initiating solution, and generating a path in the neighbourhood space that leads toward another solution, called the guiding solution. Note that the initiating and the guiding solutions represent the starting and the ending points of the path. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions (Laguna & Marti 1999). A crucial difference between evolutionary algorithms, such as genetic algorithms, and path relinking is that the former uses a factor of randomness to create offspring from parent solutions, whereas the latter uses systematic and deterministic rules to combine elite solutions. The main principle of its deterministic behaviour is the gradual introduction of attributes from the guiding solution to intermediate solutions. These attributes should have fewer characteristics from the initial solution and more characteristics from the guiding solution as the search moves along the path (Ho & Gendreau 2006). Path relinking has been considered to be particularly appropriate to tabu search, as it allows for ‘tunnelling’ through infeasible regions formed from the tabu list (Glover 1997). Figure 1.1 demonstrates our interpretation to the path relinking process in the context of graph drawing. Initial and guiding solutions are two different layouts chosen during the execution of tabu search algorithm. Then, the path relinking procedure performs a “tunnelling” operation in the solution space between the two solutions.

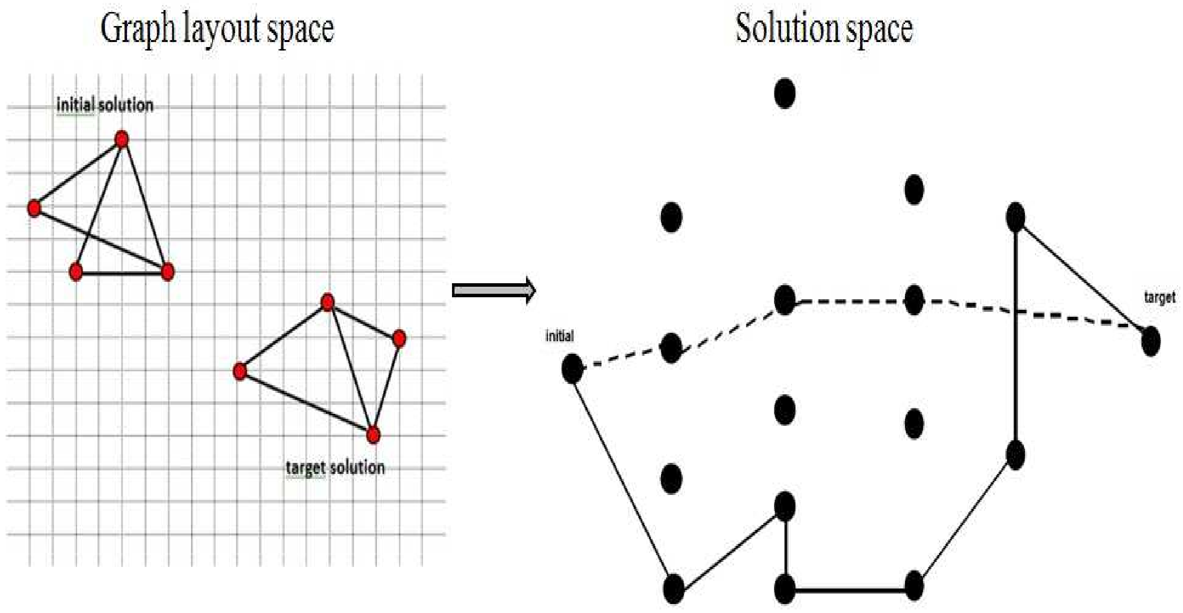


Figure 1.1 Path relinking tunnels through areas between initial and guiding graph layout solutions

In order to reach our objective, we had to implement and evaluate our method against the two commonly used alternative neighbourhood search-based methods for graph drawing. The comparison was based on the three types of evaluations that were carried out: finding the best layout that can be achieved; how long it takes to draw a graph to a particular level of quality; and how good the quality of the graph is after a fixed optimisation time.

1.2 Contributions

The major contribution of this thesis is proposing a novel neighbourhood search-based graph drawing algorithm that improves the current state of the art in neighbourhood search for drawing general graph layouts with undirected straight lines based on a weighted sum multi-criteria optimisation. This contribution can be broken down into several smaller contributions:

1. The development of a piece of software that can be used for testing the methods on random graph layouts based on Erdos-Renyi model (Erdos & Rényi 1960; Daudin et al. 2008), and real world datasets. It also allows the user to control the values of the parameters for each method and the weight of each aesthetic criterion in the fitness function. This section of the work is described in Chapter 3.

2. The implementation of a novel neighbourhood search-based method that improves the current state of the art in neighbourhood search methods. We started with proposing a tabu search-based approach for graph drawing and we compared it with hill climbing and simulated annealing. The method searches for the best positions of the nodes that minimise the value of the fitness function, and draws a nice graph layout accordingly. Tabu search forbids moves that have been previously examined which may be considered poor potential solutions, making it a more effective layout method than hill climbing. We show that tabu search alone outperforms hill climbing, but not simulated annealing. This section of work is described in Chapters 4 and 5, and a description of an initial version was published (Dib & Rodgers 2014).

3. An improvement to the proposed method by combining it with path relinking which outperformed simulated annealing. The tabu search algorithm outperforms hill climbing in minimising the value of the fitness function and the number of evaluated solutions used to draw a graph layout. The addition of applying path relinking within the tabu search procedure speeds up the identification of good solutions and outperforms simulated annealing by producing graph layouts with better values of the fitness function. We also demonstrate that when targeting a particular value of a fitness function, the combination of tabu search and path relinking achieves the goal in a smaller number of evaluated solutions. Note that the criteria of comparisons between

the methods are based on the number of evaluated solutions required to draw a layout (as that is a machine independent criterion), and the value of the fitness function of that layout. In addition, we present an execution time comparison when we test the scalability of the methods. We use the execution time to formulate a realistic conclusion of the run time for applying the methods. Statistical significance tests and effect size measurements that confirm the results of our experiments are also conducted. Finally, we show that similar results can be produced in a real world setting by testing our method against a standard public graph dataset. This section of the work is described in Chapters 6 and 7, and it was published (Dib & Rodgers 2018).

1.3 Publications

The following is a list of publications along with their related chapters and contributions:

- Refereed journal article: *A tabu search-based approach for graph layout,* in the Journal of Visual Languages and Computing (JVLC) (Dib & Rodgers 2014). This paper was accepted at the 2014 international workshop on Visual Languages and Computing (VLC) as part of the 2014 international conference on Distributed Multimedia Systems (DMS). Papers accepted at the DMS were published in a special issue of the JVLC after an additional round of reviews. The work in this paper appears in Chapters 3, 4, and 5 (Contributions 1 and 2). My contribution in this article included: conceiving and proposing the graph drawing algorithm, implementing the algorithm, performing the experimentation, and writing-up the article.

- Refereed journal article: *Graph drawing using tabu search coupled with path relinking*, in PLoS ONE (Dib & Rodgers 2018). The work in this paper appears in Chapters 3, 6, and 7 (Contributions 1 and 3). My contribution in this article included: conceiving and proposing the graph drawing algorithm, implementing the algorithm, performing the experimentation, and writing-up the article.

1.4 Software Implementation and Online Resources

In order to test the performance of all the drawing methods on graph layouts with multiple metrics and in terms of time and quality, we implemented our own software visualisation tool using Java. The tool consists of a graphical interface that includes a drop-down menu with different options, where nodes and edges can be created and manipulated. The tool provides the feature of importing/exporting graphs from/to text files. Moreover, the feature of generating random connected graphs, based on Erdos-Renyi model, is provided using the built-in random function in Java. The graph generator accepts the number of nodes and the density of the graph to be generated and then the generator produces a random connected graph accordingly. The user can select the preferred neighbourhood search-based technique to apply on the imported graph layout.

In multi-criteria graph drawing, the weight of each metric could change for each layout as it depends on the metric in which the user prefers to focus on. Therefore, we facilitate the parameter tuning process for each method and the selection of weight for each aesthetic metric by providing a smaller frame that contains text fields where these values can be controlled. The frame also shows the value of each individual aesthetic measure after optimisation, in addition to the value of the weighted sum of the fitness function.

A detailed description of our visualisation tool is provided in Chapter 3 of this thesis. The code and data related to this research can be accessed at the Dryad Digital Repository: https://doi.org/10.5061/dryad.k082rv8.

1.5 Overview of Chapters

This thesis is divided into several chapters, as described below:

Chapter 2

It includes a literature review on graph drawing and the aesthetic of graph layouts. A number of different graph drawing techniques, such as force-directed approaches and search-based approaches (including population-based and neighbourhood search-based approaches), that were introduced in the literature are highlighted. It also introduces the background of tabu search and path relinking as search-based techniques and it shows their effectiveness in many graph applications and multi-criteria optimisation problems.

Chapter 3

It describes the features of our visualisation tool along with the operations that can be performed in order to test our graph drawing algorithms and perform all the experiments conducted in this research.

Chapter 4

It describes the basic neighbourhood search-based graph drawing algorithms along with the parameters' tuning process for hill climbing and simulated annealing, followed by our tabu search-based approach for drawing general graph layouts with straight lines that have multiple aesthetic criteria which are used in a weighted fitness function to measure the quality of the graph layout. The process that we used for normalising the values of each aesthetic measure is also described.

Chapter 5

It demonstrates the experimental results of applying hill climbing, simulated annealing, and our graph drawing version of tabu search on random graph datasets and real world graph datasets. It also shows our analysis and conclusions to the results.

Chapter 6

It describes the proposed algorithm by showing the process of integrating path relinking within tabu search along with the calibration of parameters. The reason behind choosing path relinking is clarified. It also discusses different variations of path relinking that can improve the performance of the algorithm.

Chapter 7

It demonstrates the effect of coupling the tabu search graph drawing algorithm with path relinking. A comparison with simulated annealing is made by applying the methods on random and real world graph datasets. It also shows the process we followed for analysing the performance of our method and for testing its scalability.

Chapter 8

It summarises the objectives, contributions, and findings of this thesis, and covers a number of ideas for future research in this area.

1.6 Summary

In this chapter, we described the motivation behind conducting this research along with the objectives that we needed to achieve. We also demonstrated our contributions and their related publications. A brief description of our visualisation tool that had been used in our experiments was shown. In the next chapter, we review the background material relevant to this thesis.

Chapter 2 Background and Related Work

This chapter describes the background material relevant to the research in this thesis. It starts with introducing the concept of graph drawing and diagram visualisation. Then, it describes the aesthetics of graphs and their importance in improving the human understanding of graph layouts. Also, a number of different graph drawing techniques, such as force-directed approaches and search-based approaches (including population-based and neighbourhood search-based approaches) that have been introduced in the literature, are highlighted. Lastly, it introduces the background of tabu search and path relinking as search-based techniques and it shows their effectiveness in many graph applications and multi-criteria optimisation problems.

2.1 Introduction

Graphs are commonly used data structures in many fields of computer science, such as state graphs, networks, data-flow diagrams, and entity-relationships diagrams. A graph can be defined as a set of nodes and a set of edges. Two nodes are said to be adjacent if they are connected by an edge. The edge connecting two nodes represents the existence of a relationship between them. This relationship could be symmetric or asymmetric based on the type of the edge whether it is undirected or directed. In this research, we focus on undirected graphs.

In this data structure, relationships can be represented in a tabular form using an adjacency matrix or adjacency lists. Visualising a graph can help gain a better understanding of those relationships. The way of drawing a graph has a significant impact on how humans understand the relationships between the nodes of the graphs. Therefore, the layout and the arrangement of the nodes highly affect the interpretation and the readability of the relationships in the graph (Purchase et al. 1996; Purchase 1997). In order to lay out a graph automatically, graph drawing algorithms are required to rearrange the nodes in a way that emphasises the relationships without misleading the user of the generated layout.

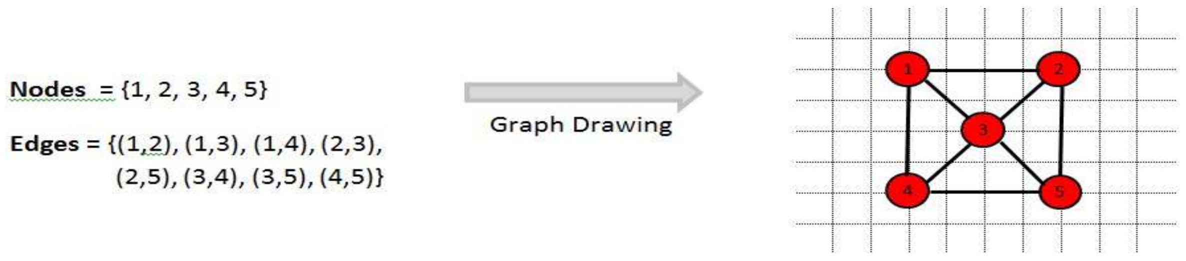
We begin with definitions and notations of the terms for graph concepts that will be used throughout this work.

2.2 Definitions

A graph is denoted by G=(V,E) where V is a set of nodes and E ⊆ (V×V) is a set of edges. The number of nodes and the number of edges are denoted by |V| and |E|, respectively. For the purpose of this work, it is assumed that G is connected, undirected, and simple (i.e. has neither self-loop edges nor multi-edges between any two nodes). When embedded in the plane, the nodes have *x* and *y* coordinates (*x*,*y*), and the edges are straight lines joining the coordinates of the two nodes. d_{ij} represents the Euclidean distance between two nodes *i* and *j*, whereas *deg*(*v*) denotes the degree of a node *v* (i.e. the number of edges incident to *v*). A layout of a graph G is a bijective function that maps each node *v* to a distinct point *layout[v]*, and each edge (*u*, *v*) to a distinct edge with endpoints *layout[u]* and *layout[v]*.

2.3 Overview of Graph Drawing

Graph drawing is the process of turning an abstract graph into a graph with an embedding in the plane that is called a graph layout. A sample graph layout is shown in Figure 2.1. This representation should aid the analysis and understanding of the graph. Graph drawing is an area of computer science which combines graph theory and information visualisation. Graph layouts are not only used in the field of computer science. For example, they are used in: physics and chemistry in modelling the interaction between particles, social sciences in drawing graphs of group interaction, and electrical engineering in representing circuits. However, the drawing process is a significant challenge. Firstly, it depends on what we refer to as a nice graph and secondly, it depends on the efficiency of its automated implementation. Many sophisticated algorithms were proposed to address the problem of displaying complicated graphs of high complexity in structure and size (Huang et al. 2007; Dogrusoz et al. 2007; Dogrusoz et al. 2009).



Nodes = {1, 2, 3, 4, 5}
Edges = {(1,2), (1,3), (1,4), (2,3), (2,5), (3,4), (3,5), (4,5)}
Graph Drawing
1
2
3
4
5

Figure 2.1 Sample graph layout

Many graph drawing algorithms were implemented taking into account one or more aesthetic criteria that would increase the readability of the drawing (di Battista et al. 1999). There are several multi-criteria approaches to layout graphs discussed in the literature, a number of which are investigated later in this work. They are based on explicit cost functions that combine the explicit measurements of graph quality. Generally, all graph drawing approaches aim to enhance the readability of the graph and to convey the information that the graph contains. In some approaches, the positions of the nodes are restricted, e.g. they are placed on grid points (Batini et al. 1986; Tamassia et al. 1988), concentric circles (Carpano 1980), or parallel lines (Sugiyama et al. 1981). The edges, on the other hand, can be drawn as straight lines, curves, or polygonal lines.

Graph layouts depend on different aesthetic qualities that could aid a better understanding of graphs and consequently build more effective systems. Purchase (1997) performed experiments on general graphs which showed a strong evidence to support minimising edge crossings for increasing the readability of a graph layout in addition to an effect of maximising the minimum angles between two incident edges to a single node. Additional aesthetics were discussed in Purchase (2002). However, aesthetic selection is a subjective process that makes the field of graph drawing more challenging. In fact, Blythe et al. (1995) asserted that there is no best way to draw a graph and that a layout simply depends on the criteria of the graph we wish to highlight. These might include specific aspects of the structure of the graph itself, particular measures of centrality, or certain attributes of the nodes or edges (Gibson et al. 2013). For example, Figure 2.2 represents two symmetric graph layouts. But the users find the layout on the left easier to understand than the one on the right although the latter has no edge crossings (Gibson et al. 2013).

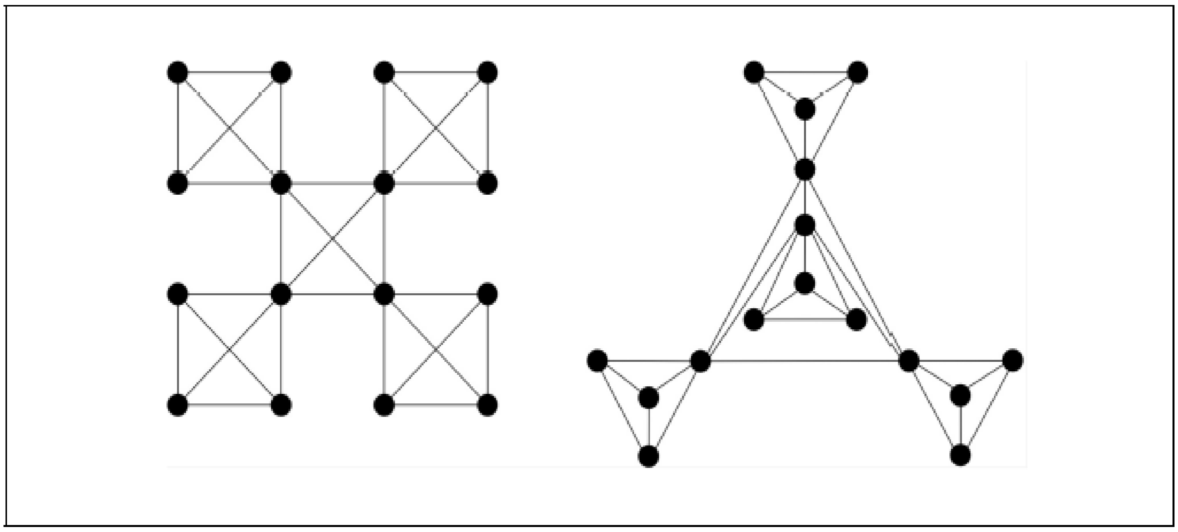


Figure 2.2 Two symmetric layouts for the same graph (Kamada & Kawai 1989)

In the next section, we describe the most commonly used aesthetic criteria of graphs discussed in the literature that have an effect on the readability of the graph.

2.4 Graph Drawing Aesthetics

Graph drawing aesthetics are quality measures that determine the readability and usability of graphs. A good layout can clearly deliver information, whereas a poor layout can mislead (Purchase et al. 1996). Graph layout algorithms typically conform to one or more aesthetic criteria. Metrics are used to measure these criteria in a weighted sum to quantify the quality of the graph layout. These aesthetic metrics can be used for the definition of fitness functions for search-based techniques, such as simulated annealing and hill climbing. These criteria include edge crossings, edge length, edge bends, node-to-node and node-to-edge occlusions, graphs symmetry, the angular resolution of the incident edges, and octilinearity (edges should be drawn horizontally, vertically, or diagonally) (Eades 1984; Kosak et al. 1991; di Battista et al. 1999; Davidson & Harel 1996; Stott et al. 2011). Formal continuous metrics for measuring the aesthetic presence in a graph drawing for seven common aesthetic criteria applicable to any graph drawing of any size were presented by Purchase (2002). Metrics can be continuous or discrete. Analysing the graph layout with continuous metrics would not be considered a binary decision, but it would indicate the percentage in which the drawing conforms to the aesthetic.

An empirical study was conducted by Purchase et al. (1996) on the human understanding of the undirected graphs drawn using three commonly used graph drawing aesthetics: symmetry, minimising edge crossings, and minimising bends in polylines. The study confirmed that increasing the number of edge crossings and the number of edge bends would decrease the readability of the graph. Therefore, minimising these two aesthetics is justified. The study was unable to conduct any conclusive assessment of the effectiveness of the local symmetry hypothesis. Each aesthetic was considered separately by comparing graphs with the extremes of the same aesthetic. Further empirical tests were conducted by Purchase (1997) that resulted in showing strong evidence for minimising edge crossings and weaker evidence for minimising the number of bends and maximising perceptual symmetry. The study also concluded that maximising the orthogonal structure of the drawing and maximising the angles between incident edges appear to have little effect on understanding the graph.

In this work, a list of aesthetics for measuring a graph layout quality was determined. The list includes: nodes distribution, edge lengths, edge crossings, node-to-edge occlusions, and angular resolution. The following is a description of the metrics used to measure the quality of each aesthetic as described by Davidson and Harel (1996) and Stott et al. (2011):

a. *Node distribution (m_1)*

Spreading the nodes out evenly on the drawing space makes the graph look nice and readable. The distances between close nodes should be increased (minimising the inverse), or in other words, the nodes should not be too close to each other (see Figure 2.3). This criterion is measured using the following formula that should be minimised:

$$\sum_{i \in V} \sum_{j \in V} \frac{1}{{d_{ij}}^2} \quad where\ i \neq j$$

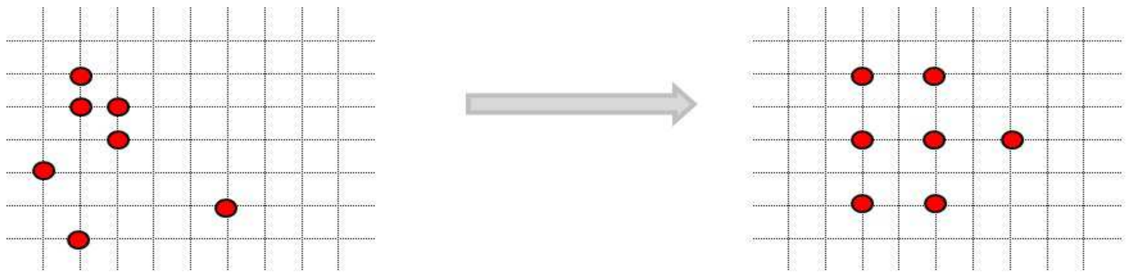


Figure 2.3 Node distribution

b. Uniform edge length (m_2)

Edges of similar lengths would make the graph look pleasant in many cases (Stott et al. 2011). The purpose of this aesthetic is to make a consistent length of all edges. A specific length (*len*) is defined, then all the edges would be adjusted in order to obtain the required length (*len*) (i.e. to penalise shorter and longer edges) using the following formula (see Figure 2.4):

$$\sum_{e \in E} (e - len)^2$$

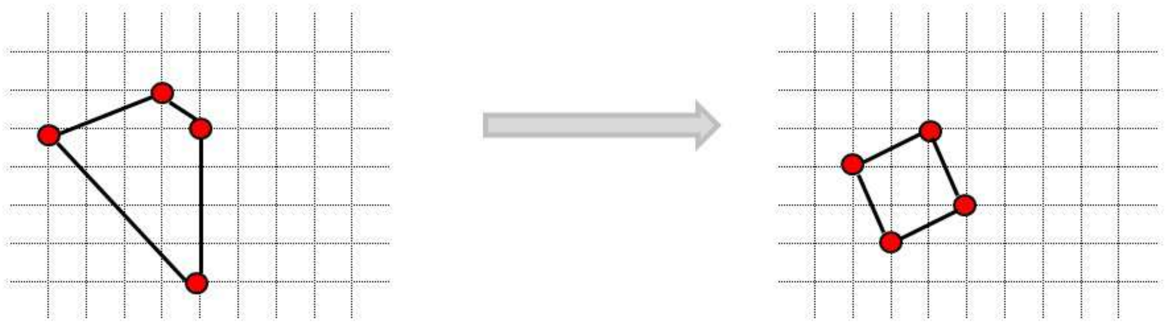


Figure 2.4 Uniform edge length

c. Edge crossings (m_3)

Planar graphs are most likely nice graphs. Minimising the number of crossing edges will lead to a planar graph layout (if the graph under study is planar). Algorithms for producing crossing-free graphs do exist (Eades & Wormald 1994; Leighton & Rao 1999; Chuzhoy 2011). However, we would like to retain the other criteria as well. Therefore, in this measure, we focus on finding the number of edge intersections and we try to minimise that number (see Figure 2.5).

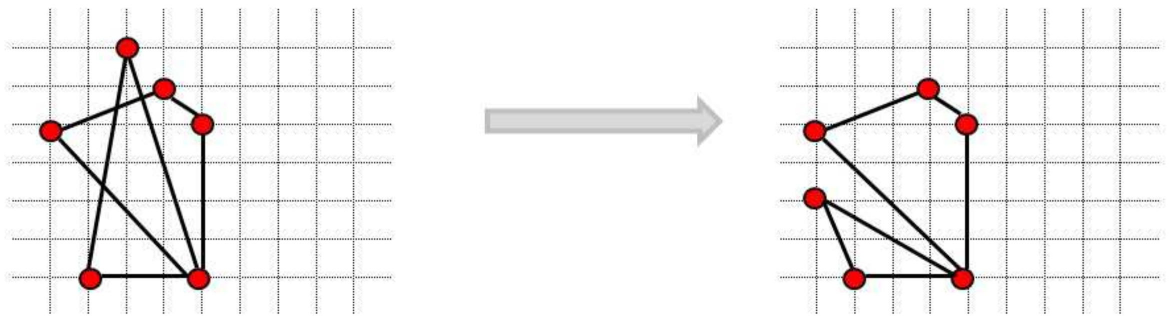


Figure 2.5 Edge crossings

d. Node-edge occlusion (m_4)

The edge crossings criterion does not take into consideration the nodes that can be positioned on edges. Therefore, the distances between the nodes and edges should be taken into account (see Figure 2.6). These distances should be increased (minimising the inverse) according to the following formula:

$$\sum_{i \in V} \sum_{e \in E} \frac{1}{{d_{ie}}^2}$$

where d_{ie} is the Euclidean distance between node i and the closest point on edge e (note that i does not equal to any of the end points of edge e).

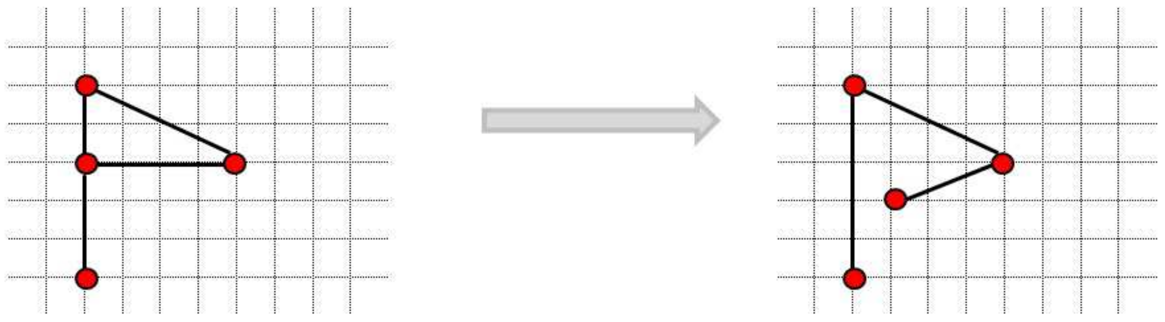


Figure 2.6 Node-edge occlusion

e. Angular resolution (m_5)

In order to have a graph where edges with a common node are not too close to each other, we should increase the distance between the incident edges (see Figure 2.7). This measure is computed as follows:

$$\sum_{v \in V} \sum_{\{e_1,e_2\} \in E} \left| \frac{2\pi}{deg(v)} - \theta(e1, e2) \right|$$

where $\theta(e_1,e_2)$ is the angle in radians between two adjacent edges e_1 and e_2 incident to node v.

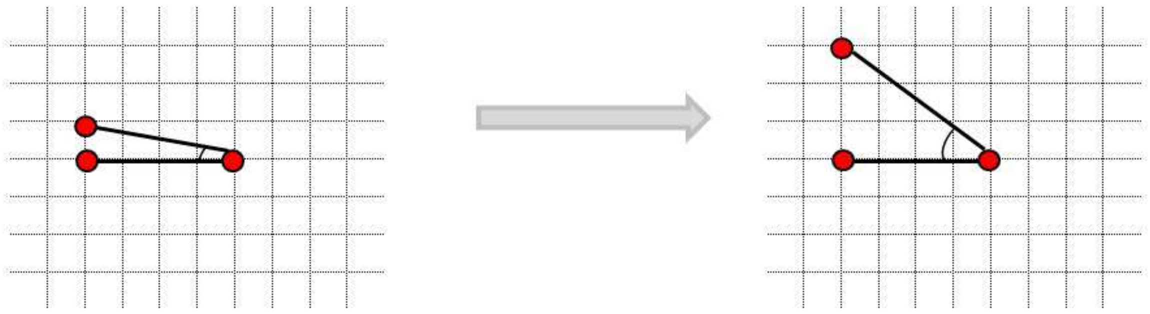


Figure 2.7 Angular resolution

Metrics are usually defined objectively, and they are not intended to take human value judgements based on the perception of what appears to be a good graph layout into account. However, the validity of human value judgements requires more extensive empirical studies and should not be based on personal opinions (Purchase 2002).

All these metrics contribute in the graph quality weighted sum fitness function that could be computed as follows (Davidson & Harel 1996):

$$fitness = w_1*m_1 + w_2*m_2 + w_3*m_3 + w_4*m_4 + w_5*m_5$$

where w_i and m_i are the weight and the measure for criterion i respectively. Note that, increasing the value of w_i compared to other weights would give the corresponding criterion a higher priority when optimising the graph given that the measures are normalised.

2.5 Graph Drawing Approaches

Graph drawing is a difficult problem (Garey & Johnson 1983; Miller & Orlin 1985). Therefore, acceptable heuristics are generally required to find good drawings and layouts of graphs. Several graph drawing techniques work better on graphs belonging to specific classes (di Battista et al. 1999). Next, we present two main divisions of graph drawing algorithms: *force-directed approaches* and *search-based approaches*.

2.5.1 Force-directed Approaches

Force-directed algorithms use a physical analogy to model the graph layout problem. They represent the graph as a system of bodies (nodes) with forces (edges) acting between the bodies. The algorithms seek for a configuration (layout) of the bodies, where each body has a position such that the sum of forces on each body is zero (i.e. a configuration where the energy is locally minimal). As forces tend to apply equally for all nodes, graphs drawn with these algorithms tend to have consistent edge lengths.

Force-directed approaches are commonly used because they are easy to understand and relatively easy to code. Moreover, the experiments with force-directed approaches show that they often give good results and can produce nice layouts of some of the well-known graphs in graph theory (di Battista et al. 1999).

A force-directed approach consists of two components: the model and the algorithm. The model is a force or an energy model that measures the goodness of a graph layout. It is usually a quantification of the graph layout aesthetics. The algorithm, on the other hand, is an optimisation technique for finding an equilibrium configuration of the system (i.e. locally optimal layout).

Many force-directed algorithms have been proposed and tested. They differ in both the formulation of the force or energy model, and in the optimisation technique used to find an optimal energy configuration. The spring embedder (Eades 1984) uses a model of springs and electrical forces. Nodes are modelled as equally charged rings that repel each other (repulsive force), and edges are modelled as springs attached to the rings (attractive force). The force of the spring that is calculated in terms of the logarithm of the distance between the nodes, makes connected nodes attract each other. A repulsive force is also applied using an inverse square law. These two forces contribute to drawing edges of roughly similar length and ensuring that non-adjacent nodes are kept apart.

The algorithm works as follows: Firstly, the rings are placed in random locations forming an initial layout. Secondly, a process for calculating the force on each ring and moving the rings accordingly is repeated several times until the spring force on the rings moves the system to a minimal energy state. Note that the experiments indicated that repeating the process 100 times is enough in most of the cases to reach a minimal energy state. Calculating the force on each node takes time proportional to the square of the number of nodes in the graph: each iteration of the spring embedder runs with time $O(n^2)$. Figure 2.8 and Figure 2.9, as described by Eades (1984), show how the spring embedder lays out the complete graph with six nodes, K_6.

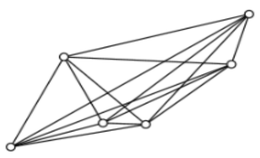


Figure 2.8 Randomised graph of a complete graph with 6 nodes

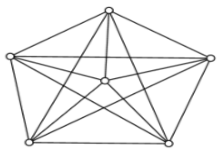


Figure 2.9 Embedded graph of a complete graph with 6 nodes using the basic spring embedder

The algorithm showed an acceptable running time for graphs with a small number of nodes. However, there are some classes of graphs for which the algorithm does not produce a good layout, such as: dense graphs or graphs with dense sub-graphs, and graphs with a small number of bridges (Eades 1984).

The spring embedder model has been modified by eliminating the electrical charges and instead associating a spring with every pair of nodes rather than just with the edges (Kamada & Kawai 1989). This modified model has been conceptualised in terms of energy rather than forces and it has been used for drawing undirected graphs and weighted graphs for human understanding. This approach uses the relation between the graph theoretic distance and the geometric Euclidean distance between nodes to produce good layouts.

The algorithm transforms the graph layout problem into a virtual dynamic system, such that every two nodes are connected by a virtual spring of desirable distance. Hence, the optimal layout of the graph is the state in which the total spring energy is minimal. The total balance condition is computed as the square summation of the differences between the desirable distances and the geometric distances for all pairs of nodes.

This approach differs from the one presented by Eades (1984) in its optimisation algorithm. Instead of moving all the nodes at once, the algorithm moves only one node in the drawing per iteration. In each iteration, the algorithm moves the node experiencing the greatest net energy, by solving partial derivatives of the energy function, to a point of locally minimal energy using a variation of the Newton-Raphson method (Rowe et al. 1987).

The algorithm works particularly well for symmetric graphs and is relatively good at minimising edge crossings. The main disadvantage of this approach is its time complexity. The model requires a pre-processing step that computes the shortest paths for every pair of nodes. The time complexity of this step is $O(n^3)$ which makes this approach impractical for large graphs (Rowe et al. 1987).

An improved algorithm for the spring embedder model was presented by Fruchterman & Reingold (1991). The main goals for the proposed method were speed and simplicity. Many graphs were drawn in less than a second, but measures were taken to restrict the graphs to a maximum of 100 nodes. The method strives for uniform edge lengths, and it also performs well in terms of distributing nodes evenly and reflecting symmetry.

In Fruchterman & Reingold (1991), a better cooling schedule could have significantly improved the algorithm. Therefore, an enhancement was made by Frick et al. (1995) by proposing an adaptive schedule with local and global temperatures and the algorithm is known by the Graph EMbedder algorithm (GEM). The algorithm was able to match or even improve the quality of the results obtained by other widely used implementations while running consistently faster than them. The algorithm was tested to produce graph layouts with evenly distributed nodes and edges with equal lengths. Although the GEM was not designed to explicitly minimise edge crossings, it can often avoid crossings.

Van Ham & Van Wijk (2004) proposed a new force model with continuous visual abstraction that combines both explicit clustering and visual clustering for drawing graph layouts that better reflect the natural cluster structure of small world graphs. The model uses the concept of force annealing, which combines force-directed algorithms that model a graph as a physical system, then it attempts to find positions for all nodes such that the total energy in the model is minimal, using a method of optimisation that starts with a random configuration. This method showed better results compared with conventional force-directed approaches when being applied on a cross referenced database of 500 artists (Van Ham & Van Wijk 2004). Force annealing models were also used in other applications such as preventing nodes from crossing edges (Simonetto et al. 2011).

Maaten & Hinton (2008) presented a t-distributed Stochastic Neighbour Embedding (t-SNE) which is a non-linear dimensionality reduction visualisation technique used to visualise high-dimensional data by assigning each data-point a location in a two or three dimensional map. The visualisations produced by t-SNE were better than those produced by other non-parametric visualisation techniques such as Isomap (Balasubramanian & Schwartz 2002).

Graph drawing with force-directed approaches that are based on virtual physical models is still considered a hot topic that has been addressed in many recent research studies (Noack 2007; Gansner et al. 2013; Jacomy et al. 2014; Ortmann et al. 2016).

Force-directed approaches typically produce aesthetically pleasing layouts. They are fast when being applied on small and medium size graphs, where the speed is highly beneficial for use in interactive systems, but they are often unable to escape local optima due to their physical model. These approaches are computationally expensive to find a minimum energy state using general energy functions. A disadvantage of these techniques is that new criteria can only be enforced by applying additional forces to the nodes causing them to move differently. This makes it very difficult to strongly enforce additional criteria as nodes are moved by summing all their forces in each iteration. Hence, the resulting composite force satisfies none of the applied criteria, and nodes are moved to non-optimal positions. Furthermore, force-directed approaches are usually selected to draw graph layouts when we want to obtain uniform edge lengths and show symmetries in the graph (Eades 1984;

Fruchterman & Reingold 1991). However, these forces may introduce a lot of edge crossings which is an aesthetic measure that cannot be turned into a force (Bertault 1999). General search-based approaches, such as simulated annealing, genetic algorithms, and hill climbing are more favourable techniques for general and discrete cost functions.

2.5.2 Search-based Approaches

The graph layout problem can also be modelled as an optimisation problem. Unlike force-directed approaches, where aesthetics can only be indirectly coded; search-based approaches have the advantage of allowing tuneable combinations of metrics to meet user preferences. Here, layout involves minima of the fitness measure that represents the desired graph aesthetics. The spring embedder approaches, described previously, mainly focus on distributing nodes and edge lengths. Both criteria were measured using a simple and continuous function of the locations of the nodes. However, many of the important aesthetic criteria, such as the minimisation of the number of edge crossings, are not continuous. Therefore, we can broaden the set of graph aesthetics by directly measuring them in the layout.

When an algorithm attempts to draw a graph layout according to several graph aesthetic criteria, some of these criteria might conflict with each other. Hence, we can use a fitness function that linearly combines a number of measures. The weighted sum method allows the multi-objective optimisation problem to be transformed as a single objective optimisation function that is constructed as a sum of objective functions f_i (measures) multiplied by weighting coefficients w_i (Grodzevich & Romanko 2006). The problem is formulated as follows:

$$min \sum_{i=1}^{k} w_i f_i \,, \qquad where\ w_i \geq 0, \forall i = 1, \dots , k.$$

The functions might include both continuous functions (like those used in the spring embedder approaches) and discrete functions. In this way, the fitness function would measure the quality of the graph layout (di Battista et al. 1999).

The problem with using general fitness functions is that it might be computationally expensive to find a minimum fitness value. Since the overall fitness function could include both continuous and discrete functions, some general search-based approaches, such as population-based methods including genetic algorithms, and neighbourhood search-based methods such as simulated annealing, and hill climbing, were used in order to find a minimum fitness value. However, they are computationally expensive and not suitable for interactive systems (di Battista et al. 1999). The main difference between these methods is the number of solutions examined at the same time. While neighbourhood search methods work on a single solution at a time, population-based methods evolve a set of points in the search space (Blum & Roli 2003). Parameter tuning for all these methods plays an important role in increasing their efficiency.

2.5.3 Multi-level Approaches

Multi-level graph drawing methods are frequently applied to clustered graphs (i.e. graphs with recursive clustering structures over the nodes) (Eades & Feng 1996). The technique repeatedly groups the nodes to form clusters which in turn are used to define a new graph. The coarsest graph is then partitioned where each partition is refined on all the graphs starting from the coarsest and ending with the original (Walshaw, 2000). This type of graph is commonly visualised at multiple abstraction levels such as a three dimensional drawing where each level is drawn on a plane at different *z*-coordinate while the clustering structure is drawn as a tree in three dimensions. This representation preserves the mental map between abstraction levels as it gives a better visualisation of the graph at different depth of abstractions and tracks the abstractions from one level to another (Eades & Feng 1996). Walshaw (2000) proposed a fast multi-level algorithm that outpeformed conventional force-directed placement and spring embedder algorithms.

Hachul & Jünger (2004) presented a fast force-directed method that is based on a combination of a multi-level scheme and a startegy for approximating the repulsive forces in the system. The algorithm managed to visualise the structures of large graphs (with up to 100000 nodes) that were challenging to visualise with some other methods.

Archambault et al. (2007) proposed a multi-level framework to draw undirected graphs based on the topological features they contain. It was the first multi-level approach that partitioned the graph into topological features. It contained a stage that reduced the number of node-edge overlaps and edges crossings and another stage to eliminate all node-node overlaps. The algorithm was compared against four other multi-level algorithms on a variety of datasets and it demonstrated improvements in terms of speed and visual quality.

2.6 Population-based Methods

Population-based methods are well-known searching methods that perform search processes that demonstrate the evolution of a set of points in the search space (Blum & Roli 2003). They provide a natural way for the exploration of the search space. The performance of these methods is strongly dependent on the way of manipulating the population. Genetic algorithms and ant colony optimisation are two popular stochastic methods which belong to this category. Finding near-optimal solutions with these methods is secured, however, a global convergence is not always guaranteed.

2.6.1 Genetic Algorithms

The genetic algorithms approach was applied to the graph layout problem as it is considered to be a good global optimiser for many optimisation problems. Genetic algorithms are stochastic global search methods that work with a population of candidate solutions and try to optimise these by means of three basic principles: selection, recombination, and mutation. The initial population is randomly chosen. Then, in every subsequent generation, new candidate solutions are produced by selecting two solutions, with higher probability of selection for better solutions, recombining parts of these solutions to form one or two offspring, and mutating the resulting offspring. Finally, the offspring is inserted into the population and the worst solution is deleted (Dorigo & Di Caro 1999).

Genetic algorithms have been successfully adapted in many single criterion and multi-criteria optimisation problems (Fonseca & Fleming 1993; Murata et al. 1996; Konak et al. 2006; Coello et al. 2006). This search-based technique has also targeted the graph drawing problem. Kosak et al. (1991) and Kosak et al. (1994) proposed a genetic algorithm-based approach for drawing graphs under a number of visual constraints. The proposed algorithm produces graphs with good quality in addition to its flexibility. It can be easily adapted to take new layout aesthetics into account. However, the major problem in this algorithm is its slow rate of convergence. It initially makes rapid progress towards a solution, but then it converges very slowly to a global optimum (or at least to a good local one).

A genetic algorithm-based approach that minimises the number of edge crossings in bipartite graphs, when the order of the nodes in one of the node subsets is fixed, was proposed in Mäkinen & Sieranta (1994). The experimental results show that the proposed algorithm outperforms some well-known heuristics that were previously applied on the bipartite graph drawing problem, such as the barycentre heuristic and the median heuristic, especially when applied on sparse graphs.

Branke et al. (1996) presented a genetic algorithm with a local fine tuner, based on the spring algorithm, for the drawing of undirected graphs with straight-line edges. According to some preliminary results, the algorithm shows its ability to produce layouts with a minimal number of edge crossings on all tested graphs. The algorithm benefits from the combination of the genetic algorithm and the spring algorithm to produce good layouts for a large class of graphs with implicit symmetry, similar spring lengths, and even distribution of nodes. Varying the weights of the criteria in the fitness function gives some control over the final appearance of the graph layout.

The layouts found by the algorithm have good general structures, but they require some fine tuning. Moreover, the comparatively long running time of the algorithm is its main disadvantage. One reason for the high time complexity of the algorithm comes from the crossover operator that was used to solve the competing conventions problem which states that a recombination of two good parents may yield a very poor offspring (Branke et al. 1996).

A similar work was introduced by Eloranta & Mäkinen (2001). This work proposed a genetic algorithm that nicely draws undirected graphs of moderate size. But the algorithm still suffers from the lack of a proper crossover operation that would speed up its computations by decreasing the number of generations needed.

Vrajitoru (2009) proposed a multi-criteria optimisation approach, using genetic algorithms, to the graph drawing problem. The study addressed the problem of building consistent graph layouts for weighted graphs following a specific geometric shape. The proposed genetic algorithm was compared to force-based algorithms. For this problem in particular, force-based algorithms were faster and more efficient in terms of performance. However, with the genetic algorithm approach, geometric shapes that present interesting geometric properties were obtained and they were visually more pleasing compared to force-based algorithms.

In summary, genetic algorithms have been successfully used for single-criterion and multi-criteria graph drawing. However, there are two major drawbacks: the slow rate of convergence to global optimum; and the long execution time due to the lack of a proper crossover operator.

2.6.2 Ant Colony

Ant colony optimisation is another population-based method that was also applied in the field of graph layout but it is not as common as genetic algorithms. This method takes inspiration from the foraging behaviour of some ant species (Dorigo et al. 2006). These ants deposit a substance on the ground to guide other members in the colony to follow a favourable path. The chemical substance trails enable ants to find short paths between their colony and food sources. The ant colony system exploits a similar technique for solving optimisation problems.

In the field of graph drawing, ant colony optimisation was applied to draw a special type of graphs related to business process diagrams (Jancauskas et al. 2012). This problem is defined as redrawing the lines that represent the sequence flow for fixed flow objects and defined sequence flow, in a pleasant layout. The problem was reformulated as a multi-criteria combinatorial optimisation problem, where aesthetic criteria, such as the length of lines and the number of line crossings and bends, were considered in a fitness function that should be minimised. The ant colony was applied on randomly generated test problems with different complexities. The experimental results showed that ant colony optimisation is a promising technique to solve this type of problem.

The automatic schematising of transport network data sets is another application where ant colony optimisation was used (Ware & Richards 2013). The problem is defined as generating an alternative network from an initial network layout by moving its vertices, reorienting edges, and increasing or decreasing the lengths of its edges. An ant colony system was implemented for the purpose of producing better results and in order to ensure a faster execution time compared to the other search-based techniques which were used for schematising transport networks. The system was tested and evaluated empirically. The results of the experiments showed that the ant colony system can be effectively used in schematising transportation maps since it outperformed previous algorithms which were applied for the same purpose, in terms of the quality of the generated maps and algorithm's execution time.

Ant colony optimisation was broadly applied to many multi-criteria optimisation problems. These problems include: the vehicle routing problem with time window constraints (Gambardella et al. 1999), the transportation problem with bi-objective combined in a weighted sum (Parragh et al. 2008), the bi-objective scheduling problem (Iredi et al. 2001), portfolio optimisation (Doerner et al. 2006), and the quadratic assignment problem (López-Ibánez et al. 2004).

In summary, the ant colony approach is not widely used in the field of graph drawing, but it showed promising results in the graph drawing applications in which it was used in. The long running time was its major disadvantage (Jancauskas et al. 2012).

2.7 Neighbourhood Search-based Methods

Unlike the population-based methods which perform searching processes that describe the evolution of a set of points in the search space, neighbourhood search-based methods work on a single solution at a time. This searching technique describes a trajectory (path) in the search space during the search process starting from a single solution (Blum & Roli 2003). Hill climbing, simulated annealing, tabu search, and path relinking, are four different optimisation techniques that go under the umbrella of neighbourhood search methods. Many graph layout algorithms in the literature used neighbourhood search-based methods, such as simulated annealing and hill climbing which are considered the most popular neighbourhood search methods. In the following subsections, we demonstrate different graph drawing and multi-criteria applications where simulated annealing and hill climbing were used. Then, we dedicate separate sections for describing tabu search and path relinking as they form the core of the algorithm proposed in this research.

2.7.1 Simulated Annealing

Simulated annealing is a search-based technique that has been widely used in a variety of optimisation problems. It is inspired by the process of cooling and freezing a metal into a crystalline structure with minimum energy. The annealing process was firstly proposed by Metropolis et al. (1953). This search-based approach models the physical process of heating a material and then slowly cooling the temperature to decrease defects that minimises the system energy. It is usually used for large-scale combinatorial optimisation problems and implemented in a way that tries to escape from a local minimum to a global minimum by applying uphill moves (moves that spoil, rather than improve, the temporary solution). These moves allow the approach to escape from a local minimal solution but with no guarantee that a global minimum can be reached eventually. This technique was applied on many single-criterion applications (Christensen et al. 1995; Ware et al. 2003) and multi-criteria applications (Ulungu et al. 1998; Suman & Kumar 2006; Smith et al. 2008; Li & Landa-Silva 2011).

The simulated annealing approach was firstly used for the graph layout problem by Davidson & Harel (1996) to draw general undirected graphs with straight line edges taking into account several drawing aesthetics: distributing nodes evenly, making edge lengths uniform, minimising edge crossings, and placing nodes not too close to edges. All these criteria were combined into a meaningful function that could be subject to the general optimisation fitness function.

The algorithm starts by choosing an initial configuration (initial graph layout) and initial temperature. Then it repeats the following steps for a fixed number of times: a new configuration is chosen from the neighbourhood of the current configuration (i.e. moving only one node in the current configuration to a new location in a range of perimeter for a circle which becomes smaller with time to get more accurate results). The fitness function of the new configuration is computed and compared to the current configuration’s fitness function. The configuration changes according to the one with the minimum value of the fitness function. Once no improvements are made, the temperature is decreased and the process is repeated until a termination rule is satisfied. Fine tuning iterations are applied on the fitness function by adding the criteria that deal with distances between the nodes and edges.

The algorithm produces nice graph layouts for small size graphs, and it also has a similar computational performance to the spring embedder approaches described earlier. However, the algorithm does not perform well for graphs of larger sizes. Another drawback of this approach is that it finds values very close to the global minimum but seldom does it detect the global minimum itself (Davidson & Harel 1996). An example of a graph layout produced by this algorithm is shown in Figure 2.10.

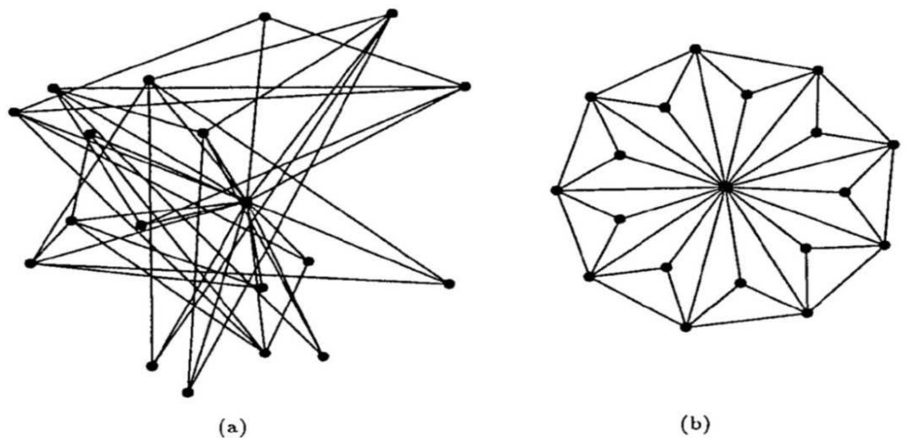


Figure 2.10 An example of a graph layout using the algorithm proposed in Davidson & Harel (1996)

An adjustment to the simulated annealing approach was made in the algorithm proposed by Brank (2004). The algorithm applies a few adjustments to the simulated annealing approach discussed in Davidson & Harel (1996) so that the fitness function can be minimised using partial differentiation and minimisation using the gradient descent. Since the fitness function is partially differentiable with respect to all its independent variables, its gradient vector can be computed. This vector, once computed on a specific node, represents the direction in which the node should move to increase the value of the fitness function. Thus, this algorithm should move the node to the opposite direction to minimise the value of the fitness function.

Applying the gradient descent technique has some challenges. For example, the fitness function should be expressed explicitly in terms of coordinates, as its derivative will be found.

Also some criteria, such as minimising edge crossings, are discontinuous and not differentiable. In Brank (2004), the gradient descent chooses a reasonable minimum length of each step to prevent the algorithm from falling into a local minimum too early. But the algorithm is still slow when being applied on larger graphs.

Lin et al. (2011) proposed an effective simulated annealing-based algorithm for drawing mental-map-preserving graphs with straight lines of general undirected graphs including six aesthetic criteria. Mental-map-preservation is about keeping the positions of the nodes as stable as possible as the graph changes. Preserving a mental map is an important aspect in graph drawing, as it allows the user to recognise the redrawn layout of the modified graph using an external visual representation instead of relying entirely on memory (Coleman & Parker 1996; Archambault & Purchase 2013). Similar to Davidson & Harel (1996), the implementation includes flexibility in terms of the weights of the graph aesthetics since the user can manually change those weights according to his/her preferences. Also, the algorithm incorporates multi-criteria simultaneously in one objective function for graph layout unlike previous works using a mental map which only included a single criterion at a time (Böhringer & Paulisch 1990; Misue et al. 1995; He & Marriott 1998). The algorithm guarantees the reduction of time required to relearn the modified drawing, but it is limited to graphs with a small size only. In addition to the experimental evaluation, the work includes a student-based questionnaire analysis for a better justification of the performance of the proposed algorithm.

In summary, simulated annealing is widely used in the field of graph drawing. It works successfully with small graphs but it is too slow when applied to large graphs. It adds an element of non-determinism in order to escape from local optima in the search space that requires a large number of iterations to obtain a good solution. Our proposed approach in this thesis should overcome this drawback by introducing a memory-based structure which excludes previously visited solutions and low quality solutions, consequently speeding up the execution time of the drawing process.

2.7.2 Hill Climbing

The second search-based approach that has been used in the field of graph drawing is hill climbing. Hill climbing is one of the simplest search-based algorithms used in the field of artificial intelligence. It is good for finding a local optimum but it is not guaranteed to find the global optimum out of all possible solutions. It works by iteratively improving a given solution that is often selected in a random way, by applying a transformation (variation) in the current solution or picking any solution in its neighbourhood. Then, the new solution is compared to the old one. If the new solution is better than the old one, the new solution substitutes the old one. This process is repeated until no more improvement is recognised on the current solution.

Hill climbing has been previously used in targeting single-criterion and multi-criteria optimisation problems (Díaz & Suárez 2001; Coello et al. 2006; Yıldız 2009; Bandyopadhyay & Saha 2012). In Flower et al. (2003), an aesthetic-based hill climbing method to draw Euler diagrams was proposed. The work concluded that it is possible to enhance the understanding of Euler diagrams with good layouts, using hill climbing, by defining a suitable set of metrics.

Hill climbing has also been used in the field of graph drawing to minimise the number of edge crossings (Rosete-Suárez et al. 1999). The experiments conducted on random graphs of different sizes showed that stochastic hill climbing outperforms efficient and popular search-based techniques, such as evolution strategies and genetic algorithms.

Stott et al. (2011) used the hill climbing approach in implementing an automatic mechanism for drawing metro maps. A good metro map layout could be evenly spaced stations, running lines at regular angles and placing labels in unambiguous locations. Therefore, Stott et al. (2011) applied multi-criteria optimisation using five different aesthetics (angular resolution, average edge length, balanced edge length incident to the same station, line straightness, and octilinearity) in a weighted sum to measure the esthetical quality of the graph. In addition to these criteria, the following rules were taken into account for each station: restricting the movement of stations to be bounded within a certain area, maintaining the relative positions of the stations, avoiding node-edge occlusions, and preserving the ordering of edges incident to a station. A hill climbing algorithm was used to reduce the value of the weighted sum and find improved map layouts. Since hill climbing does not guarantee finding the global minimum and in order to avoid local minima Stott et al. (2011) applied a clustering technique to the map. The hill climber moves both stations and clusters when finding improved layouts. The mechanism produced good map layouts and in some cases better than both published and distorted layouts. However, the performance of the algorithm was slow. To speed it up, Stott et al. (2011) suggested avoiding the comparison between nodes that are far away from each other and reusing the calculations from previous iterations.

Many graph drawing algorithms in the literature that use search-based techniques, such as simulated annealing, genetic algorithms and hill climbing, produce good layouts but they have great potential for improvement. For example, simulated annealing adds an element of non-determinism in order to escape from local minima in the search space. This would slow down the performance of the algorithm since this stochastic behaviour means that a larger number of iterations would be necessary to reach a minimum in the search space. Genetic algorithms, on the other hand, have a slower rate of convergence compared to simulated annealing and hill climbing. It initially makes rapid progress towards a solution, but then it converges very slowly to a global optimum. The main problem with hill climbing is that it gets trapped in local optima. Our proposed approach in this thesis uses an intensification technique based on a combination of tabu search and path relinking that improves the quality of solutions and speeds up the algorithm's execution time.

2.8 Tabu Search

Tabu search is a general technique that was proposed by Fred Glover (Glover 1986; Glover & Greenberg 1989; Glover 1989) for finding good solutions to combinatorial optimisation problems. Many approaches were proposed to tackle this type of problems, and the majority thereof were based on local search. In these approaches, the quality of solutions and the algorithm's computing time are dependent on the number of neighbourhood moves performed in each iteration (Gendreau & Potvin 2014).

Tabu search could be considered as a neighbourhood search method (like simulated annealing) but it takes a more aggressive approach. It proceeds on the assumption that there is no value in choosing an inferior solution unless it is necessary, as in the case of escaping from a local optimum (Lim & Chee 1991). In other words, tabu search improves the efficiency of the exploration process by keeping track of local information (like the current value of the objective function) along with some information related to the exploration process. This systematic use of memory is an essential property of this searching technique. In addition to saving the value of the best solution visited so far (like most exploration techniques), tabu search also keeps information on the itinerary through the last solutions visited. This information will be used to guide the move from solution *i* to the next solution *j* to be chosen in the set of neighbourhood solutions to *i*. The role of the memory is to restrict the choice of some subset of the neighbourhood set of node *i*, by forbidding moves to some neighbour solutions (Hertz et al. 1995). At each iteration of the exploration process, it selects the best neighbourhood solution. This is unlike hill-climbing, as it might make a down-hill move. Therefore, this technique does not run out of choices for the next move. However, this might lead in cycling by trapping the algorithm at locally optimal solutions. This problem has been resolved by introducing two structures called *Tabu lists* and *aspiration functions* which are used to keep information about past moves in order to respectively constrain and diversify the search for good solutions (Lim & Chee 1991). A flow chart that demonstrates a simple tabu search procedure is given in Figure 2.11.

The structure of tabu lists might vary from one problem to another depending on the nature of the problem. However, the most simplified form of tabu list is a linear list that stores the *k* most recent moves. The purpose of this list is to constrain the direction of search by preventing the algorithm from going back to a state that was reached previously. Using this structure might avoid being trapped in any local optimum. *Tabu conditions* are satisfied if the current move tries to undo a move previously made that is still in the tabu list. Another structure has been introduced called the aspiration function which has the ability to overrule tabu conditions by accepting some moves in the tabu list that look attractive in spite of their statuses. A tabu move is said to be attractive when applied on a current solution if it gives a better solution than the best found so far. Such a move might be accepted in spite of its status. This helps to diversify the search and encourage the exploration of new regions in the search space (Lim & Chee 1991).

The memory used in tabu search is both *explicit* and *attributive* (Glover & Laguna 1997). The explicit memory records complete solutions, typically consisting of elite solutions visited during the search. An extension of this memory records highly attractive but unexplored neighbours of elite solutions. The memorised elite solutions (or their attractive neighbours) are used to expand the local search. On the other hand, tabu search uses attributive memory for guiding purposes. This type of memory records information about solution attributes that change in moving from one solution to another.

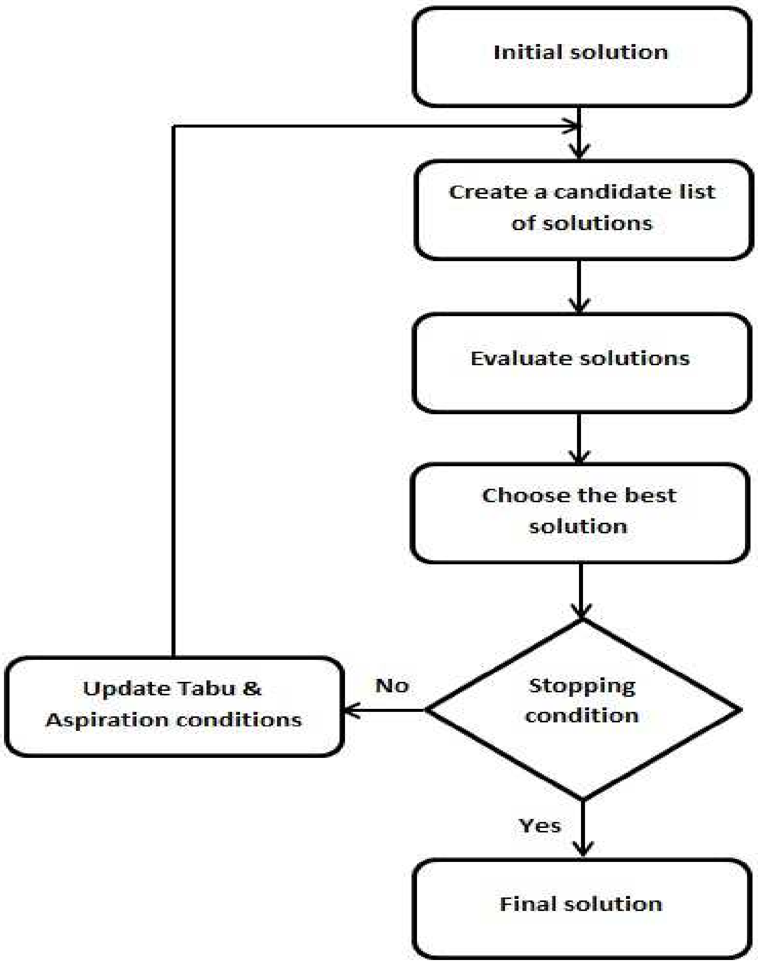


Figure 2.11 A flowchart of a simple tabu search procedure

An additional feature of tabu search is applying *intensification* and *diversification*. In the search process, it might be useful to intensify the exploration in some region because it may contain some acceptable solutions. This can be obtained by introducing a new term in the objective function that assigns a high priority to the solutions in that region that have common features with the current solution (i.e. penalise solutions far from the current one). This should be done within a limited number of iterations and then the search process should move to another region. Diversification will be responsible for moving the exploration process over different regions. Additional terms can be introduced in the objective function that penalises solutions that are close to the current one (i.e. to force the search process to jump to different regions) (Hertz et al. 1995). Algorithm 2.1 is an outline of a simple tabu search approach (Glover 1989; Glover 1990).

Tabu search can be applied on our problem for drawing a multi-criteria graph layout, but the following major points should be thoroughly investigated:

- The generation process of neighbourhood solutions.
- The structure of tabu lists and how solutions are added and deleted to/from the lists (intensification process).
- The definition of the aspiration function and how to update it (diversification process).
- The convergence properties of tabu search.

```
1.  Select an initial solution $x \in X$.
    Let $x^* = x$, where $x^*$ denotes the best solution currently found.
    Set the iteration counter $i = 0$
    Begin with an empty set of tabu moves T
2.  If $S(x) - T$ is empty, go to Step 4, where $S(x)$ is the set of all possible neighbourhood moves. Otherwise,
    set $i = i + 1$ and select $s_i \in S(x) - T$ such that $s_i(x)$ is OPTIMUM($s(x)$:$s \in S(x) - T$).
3.  Let $x = s_i(x)$.
    If $C(x) < C(x^*)$ let $x^* = x$, where C is an objective function.
4.  If a chosen number of iterations has elapsed either in total or since $x^*$ was last improved, or if $S(x) = \emptyset$
    upon reaching this step directly from Step 2, stop. Otherwise, update T by adding x if it satisfies tabu
    conditions and return to Step 2.
```

Algorithm 2.1 Simple tabu search approach (Glover 1989; Glover 1990)

Tabu search was used in solving multi-objective optimisation problems (Baykasoglu et al. 1999). The proposed algorithm was used to solve four different applications in different areas.

In every application, the algorithm’s solution was at least as good as, if not better than, the reported results using different search-based techniques.

The solution structure of tabu search, in working with more than one solution (neighbourhood solutions) at a time, enables this approach to be applied to multiple objective optimisation problems. The main stages of the basic tabu search algorithm are: *initial solution, generation of neighbours, selection* and *updating*. These stages are typical for any tabu search approach that works on single-objective optimisation problems. However, to enable the tabu search algorithm to work with more than one objective, the selection and updating stages were redefined. In addition to the tabu list, two lists were defined, the *Pareto list* and the *candidate list*. The Pareto list collects the selected non-dominated solutions found by the algorithm. The candidate list, on the other hand, collects all other non-dominated solutions that were not selected as Pareto optimal solutions in the current iteration. These solutions may become seed solutions if they maintain their non-dominated status in later iterations. The candidate list gives the opportunity to diversify the searching process (Baykasoglu et al. 1999).

Gandibleux et al. (1997) presented an algorithm based on the tabu search approach for solving multi-objective combinatorial optimisation problems, and was able to determine the efficient set of non-dominated solutions or at least a good inner approximation set of solutions.

It is always possible to use all basic tabu search techniques in multi-objective optimisation (Hansen 1997). The aspiration criterion allows the searching process to select neighbourhood solutions that can contribute to the non-dominated set even if they are the results of tabu moves, instead of only checking the best, non-tabu neighbour. Also, the neighbours resulting from tabu moves can be accepted, in some cases, as best neighbours.

In basic tabu search, whenever there is a badly connecting neighbourhood function or when the neighbourhood function induces wide valleys in the objective space, it might be needed once in a while to sample new solutions in order to be able to search the whole feasible set. This can be done by creating new, randomly generated solutions instead of duplicating existing solutions. But it can be more effective, in a systematic or probabilistic fashion, to use more than one neighbourhood function so that these connect the whole feasible set (Hansen 1997).

Moreover, the neighbourhood function might lead to generating many neighbours for each solution. Therefore, it is more efficient to make moves based on a probabilistic or systematic sampling of the neighbourhood, or to reduce the neighbourhood size. This would be useful in multi-objective optimisation problems because they have an *n*-dimensional objective boundary to discover and it is also time consuming to remain too long at each locality. Choosing the appropriate neighbourhood function might make it possible to locate the best neighbour without explicitly having to generate all the neighbours (Hansen 1997).

In Thakur & Dhiman (2011), it was concluded that tabu search can easily handle the complicating constraints that could be found in real-life applications. However, this searching technique might fail for two main reasons: an insufficient understanding of the basic concepts of the tabu search method besides a lack of understanding of the problem at hand. Selecting a proper search space and an effective neighbourhood strategy requires significant problem knowledge. Tabu search, like all meta-heuristic methods, needs to achieve both depth and breadth in its searching process; depth search is usually not a problem for tabu search, as it generally finds quite good solutions early in the searching process, whereas breadth search can be a critical issue. Therefore, it is extremely important to develop an effective diversification scheme.

Other research studies and applications that used tabu search as a technique for optimising problems with multi-criteria can be found in Brandao & Mercer (1997), Grandinetti et al. (2012), Cordeau & Maischberger (2012), and Escobar et al. (2013).

Tabu search was previously used in the field of graph drawing with a single criterion. An approach was proposed in Laguna et al. (1997) to minimise edge crossings in multi-layer hierarchical digraphs. The nodes of these diagraphs must lie on a set of equally spaced horizontal or vertical lines (layers) and all the edges flow in the same direction, as shown in Figure 2.12. Garey & Johnson (1983) proved that this problem is NP-hard even if the digraph has two layers only. The proposed tabu search approach searches for optimal or near-optimal orderings of a single layer in between its adjacent layers whose orderings are fixed (intensification phase). The algorithm also diversifies the search by applying an importance sampling procedure, based on the degree of each node, where layers are treated differently according to their level of importance. Then a switching procedure is performed on a randomly selected node in a certain layer (diversification phase).

Two versions of the proposed algorithm were deployed such that the first version focuses on the computational time when compared to methods based on simple ordering rules, whereas the second version tries to find high-quality solutions within a reasonable computing time. The only difference between the two versions was the termination criterion of the algorithm. The experiments were conducted on a set of 200 randomly generated graphs and the comparisons were made with effective techniques that were previously used in the field of edge crossings minimisation such as the barycentric and the semi media methods with switching. The experiments showed that the proposed tabu search approach is quite competitive in terms of computational time and it also produces graphs with better quality, although the difference becomes smaller at graphs with higher densities.

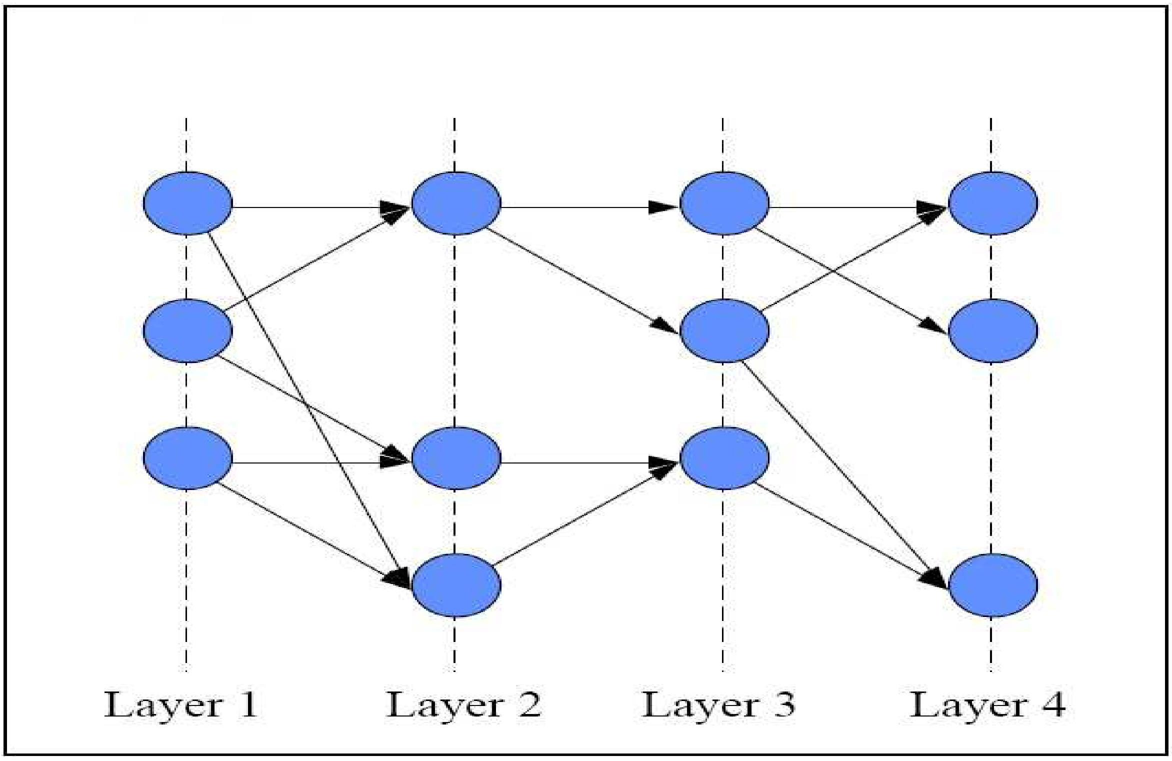


Figure 2.12 Hierarchical Diagraph (Laguna et al. 1997)

Many heuristic approaches (including tabu search) were developed to solve the bipartite drawing problem which is a special case of multi-layer hierarchical graphs (Valls et al. 1996; Marti 1998; Laguna & Marti 1999). In Martí & Laguna (2003), extensive computational experiments were conducted to explore the behaviour of the most relevant heuristic and meta-heuristic approaches developed to solve the problem of bipartite drawing, such as the Barycenter method (Sugiyama et al. 1981), the median heuristic method (Eades & Wormald 1994), Tabu search method (Marti 1998), greedy randomised adaptive search procedure (GRASP) with path relinking (Laguna & Marti 1999), and others. It is a 2-layer graph where nodes are partitioned into two disjoint subsets (left and right layers), and edges are connecting nodes between the two layers. In that work, the directions of the edges were omitted as they have no effect on crossings. A bipartite graph drawing is specified with a unique *y*-coordinate for each node, as shown in Figure 2.13. The experiments used around 3000 randomly generated graphs to compare between the methods. The research concluded that the tabu search method is more appropriate to use in solving the bipartite drawing problem as the density of graph increases with a reasonable computational time. On the other hand, the GRASP with path relinking produced better results with sparse graphs.

Tabu search has shown good results for large instances of many NP-hard problems in a reasonable amount of time (Friden et al. 1989; Hertz & De Werra 1989). It has produced comparably fast solutions in some graph theory applications, such as graph partitioning (Lim & Chee 1991; Rolland et al. 1996; Benlic & Hao 2011), graph colouring (Hertz & De Werra 1989), and weighted maximal planar graphs (Osman 2006). It has also outperformed many existing heuristics for solving the vehicle routing problem (Gendreau et al. 1994; Cordeau et al. 1997; Escobar et al. 2014).

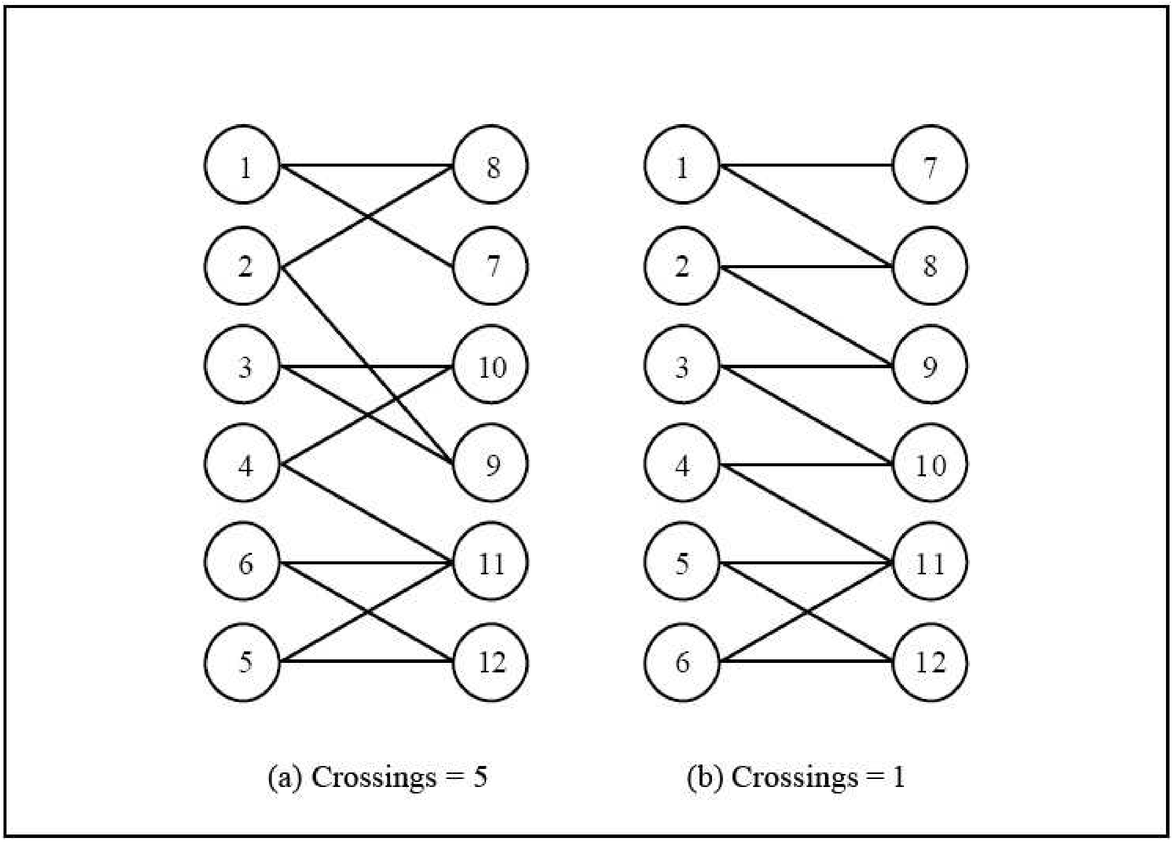


Figure 2.13 Bipartite sample drawing (Martí & Laguna 2003)

2.9 Path Relinking

Path relinking has been proposed as an approach to integrate intensification and diversification strategies (Glover & Laguna 1997; Glover et al. 2000). This approach generates new solutions by exploring paths that connect high-quality solutions (elite solutions from the *reference set*) by starting from one of these solutions, called an *initiating solution,* and generating a path in the neighbourhood space that leads toward other solutions, called *guiding solutions* where *initiating* and *guiding* solutions represent the starting and ending points of the path. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions (Laguna & Marti 1999). An illustration of a simple path is given in Figure 2.14. Unlike other evolutionary approaches, such as genetic algorithms, where randomness is a key factor in the creation of offspring from parent solutions, path relinking utilises systematic, deterministic rules for combining elite solutions. Attributes from the guiding solution are gradually introduced into the intermediate solutions, so that these solutions contain a limited number of characteristics from the initial solution and more from the guiding solution while moving along the path.

Path relinking is a fairly new approach and it has been applied in several computational problems with great success (Aiex et al. 2003; Resende & Ribeiro 2003; Ghamlouche et al. 2004; Oliveira et al. 2004; Souza et al. 2004; Aiex et al. 2005).

The following three components are crucial in the design of the path relinking technique (Ho & Gendreau 2006):

- Building the reference set,
- Choosing the initial and guiding solutions,
- Constructing a neighbourhood structure for moving along paths between initial and guiding solutions.

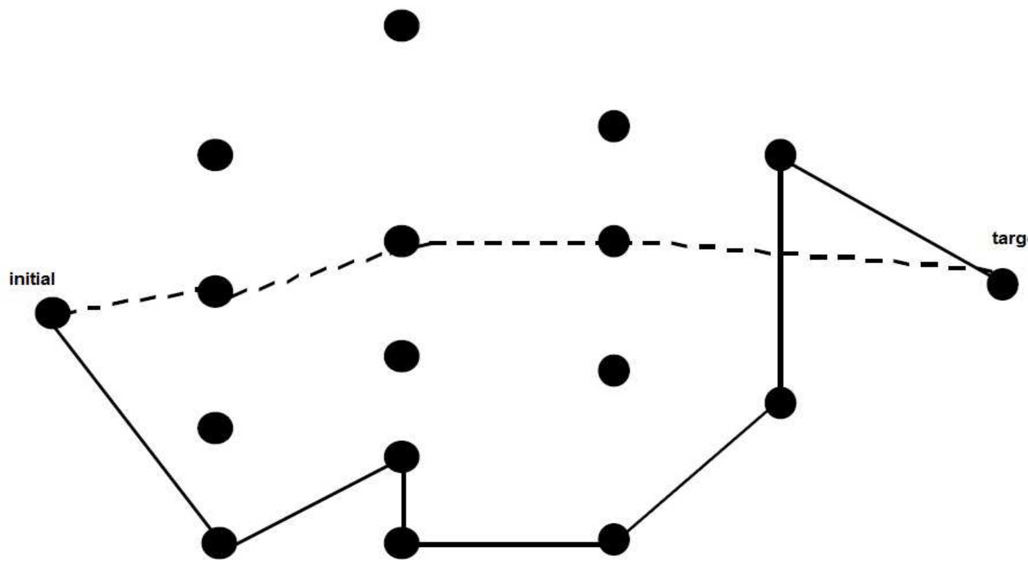


Figure 2.14 Path relinking: original path (solid line) and one possible relinked path (dotted line) in the solution space

Algorithm 2.2 shows a simple path relinking procedure that demonstrates how these components interact (Rahimi-Vahed et al. 2013).

```
1. Generate a starting set of solutions.
2. Designate a subset of solutions to be included in the reference set.
   while the cardinality of the reference list > 1
       • Select two solutions for the reference set
       • Identify the initial and guiding solutions
       • Remove the initial solution for the reference set
       • Move from the initial solution toward the guiding solution, generating intermediate solutions
       • Update the reference list
3. Verify stopping criterion: Stop or go to 1.
```

Algorithm 2.2 Simple path relinking algorithm (Rahimi-Vahed et al. 2013)

Note that using path relinking periodically in a search procedure is intended to speed up the identification of good solutions. Combining tabu search with path relinking is motivated by the desire to tunnel through blocked off areas created by the tabu solutions (Glover 1997). The proposed method in Ho & Gendreau (2006) for solving the vehicle routing problem produced computational results that show that tabu search with path relinking is able to generate better solutions than pure tabu search using considerably less computing time. Each of the three components of path relinking used in the proposed method can be implemented in different strategies as described in Ghamlouche et al. (2004). We summarise these components as follows:

i. Building the reference set

The quality of generated solutions is affected by the quality and diversity of the solutions included in the reference set. The algorithm builds the reference set during the tabu search phase and is enriched during the path relinking phase. Ghamlouche et al. (2004) proposed several strategies for building the reference set such as:

a. It is built with the solutions that at some point during tabu search become the best overall solutions (i.e. linking the overall improving solutions).

b. It contains the best local minima encountered during the tabu search phase, because usually local minimum solutions share some common characteristics with optimum solutions.

c. It is built by selecting local minimum solutions that have a better objective function value than those already in the reference set. The time aspect is introduced into the selection process since the better solutions are usually encountered when the search has been proceeding for some time. This strategy considers less local minima obtained at that stage and consequently good solutions are found early during the searching process.

d. This strategy ensures both the quality and the diversity of the solutions when building the reference set. Starting with a large set of good solutions *S*, the reference set is partially filled with the best solutions found in *S* to ensure the quality of the solutions. Then, the reference set is extended with solutions that significantly differ from those that already exist in the set.

ii. Choosing the initial and guiding solutions

The quality of the new generated solutions during the path relinking phase is highly dependent upon the initial and the guiding solutions selected from the reference set *R*. Ghamlouche et al. (2004) suggested five criteria for choosing the initial and guiding solutions:

a. The guiding and initial solutions are defined as the *best* and *worst* solutions in *R*, respectively.

b. The guiding solution is chosen to be the *best* solution in *R*, while the initial solution is the *second best* one.

c. The guiding and initial solutions are chosen *randomly* in *R*.

d. The guiding solution is chosen as the *best* solution in *R*, while the initial solution is defined as the solution with *maximum Hamming distance* from the guiding solution.

e. The guiding and initial solutions are chosen as the *most distant* solutions in *R*.

iii. *Constructing a neighbourhood structure for moving along paths between initial and guiding solutions*

The aim of the path relinking phase is to progressively introduce the attributes of the guiding solution into the solutions obtained by moving away from the initial solution. In the path relinking phase, the algorithm must ensure that a progress towards the guiding solution is made. Similarities and differences in the structure of the initial and guiding solutions should be properly identified. Identical parts of the two solutions should remain unchanged during the process. To clarify the importance of this phase, we highlight the algorithm proposed in Ho & Gendreau (2006) for solving the vehicle routing problem. Two neighbourhood methods were used. The first neighbourhood, $N_1(x)$, is made up of all the potential solutions that can be reached from x by moving customers from their current route to another while taking into account the structure of the guiding solution. The second neighbourhood, $N_2(x)$ is defined similarly as the set of all potential solutions that can be reached from x by exchanging two customers i and j between their respective routes while taking into account the structure of the guiding solution.

In Ho & Gendreau (2006), the path relinking procedure is triggered within tabu search for a predefined number of times. In each call, path relinking generates several paths with different initial and guiding solutions from the reference set such that the initial and guiding solutions are chosen according to one of the criteria described earlier. When the path is longer, the chance of producing good solutions is better. After the path relinking phase is finished, tabu search continues with the solution it had before path relinking was triggered. A calibration process was performed to adjust the frequency of triggering the path relinking procedure. This calibration process is important because if path relinking is performed too frequently, the search will tend to focus on a small portion of the search space. However, if it is performed very rarely, its impact will be negligible. Thus, it was important to find a balance between these two extremes.

A path relinking-based algorithm combined with a greedy randomised adaptive search procedure (GRASP) has been proposed to target the max-min diversity problem (Resende et al. 2010). It is an NP-hard problem, where subsets of elements should be selected from a given set such that the diversity among the selected elements is maximised. The main purpose of that work was to extensively introduce path relinking as a competitive search-based method for solving combinatorial problems. A comparison was performed with simulated annealing and tabu search that were previously proposed to target the max-min diversity problem. The results of the comparison were in favour of a variant of path relinking combined with GRASP.

The combination of tabu search and path relinking was also used to tackle the job shop scheduling problem (Peng et al. 2014). The experimental results show that this combination produces competitive results compared to state of the art algorithms for the job shop scheduling problem in the literature demonstrating its effectiveness in terms of solution quality and computational efficiency. Both techniques operate interchangeably, such that path relinking is used to generate solutions on the path from the initial solution to the guiding solution, while the purpose of tabu search is to improve the generated solution to a local optimum.

The algorithm starts by generating a random population of a predefined size of feasible solutions. Tabu search is used to optimise each solution in the population to become a local optimum. The optimisation of each solution stops when the optimal solution is found or no improvement on the best objective value is made after a given number of iterations. The reference set is updated by selecting a solution (from the initial improved population) that gives the minimum value of the objective function. Then, a pair of two solutions (initial and guiding) is randomly selected from this population. A path relinking procedure is applied on the selected solutions and returns the best solution in the path from the initial solution to the guiding solution. The returned solution is passed to a tabu search procedure with long iterations that will be compared afterwards to the solutions in the reference set and update it accordingly. The new generated solution is added to the reference set and the worst solution is removed. This process is repeated until a stopping criterion is met.

A study that shows the effect of using path relinking in the context of multi-criteria optimisation problems was presented in Martí et al. (2015). A comparison between different variants of GRASP with path relinking was made with the best methods that were previously applied on two hard bi-objective combinatorial problems. The comparison considered three different ways of implementation: firstly, each criterion is optimised independently; secondly, each criterion is optimised sequentially by alternating to guide the search; and thirdly, all criteria are combined into a single weighted objective function. The study concluded that some variants of path relinking were favoured compared to other heuristic methods.

Path relinking demonstrated efficient performance when being applied coupled with neighbourhood search-based methods and population-based methods (Ribeiro & Resende 2012). In addition to tabu search and GRASP, path relinking was successfully used in conjunction with different search-based methods, such as variable neighbourhood search, genetic algorithms, and scatter search (Canuto et al. 2001; Festa et al. 2002; Resende & Werneck 2004; Scaparra & Church 2005; Ribeiro & Vianna 2009).

Path relinking has also been applied to specialist graph drawing tasks. In Laguna & Marti (1999), path relinking was coupled with a greedy randomised adaptive search procedure (GRASP) for the problem of minimising straight-line crossings in a 2-layer graph (bipartite graph) to search for improved solutions. According to the results, the most influential factor on the performance of the algorithm was the density of the tested graphs. With reference to the experiments which were performed in Martí & Laguna (2003) to compare between 14 different heuristics, as described in the previous section, the combination of GRASP and path relinking produced better results for relatively low density graphs. The relinking process implemented in this algorithm could be summarised as follows:

During the first three iterations of the GRASP, the set of elite solutions is formed. Starting from the fourth iteration, each generated solution is considered as an initiating solution and it is subject to a relinking process by performing moves on the path from the initial solution to a randomly chosen elite solution. A move along the path is made by choosing a node from the initial solution and placing it in the position occupied by the same node in the guiding solution. Afterwards, a sequence of position exchanges of nodes, that are one position away from each other, is performed until no more improvement in the crossings minimisation is found. Once this neighbourhood process is explored, the relinking continues from the solution defined before the exchanges were performed. The relinking process stops when the initial solution matches the guiding solution. Note that, it is inefficient to apply the neighbourhood exploration process at each step of relinking since the two generated consecutive solutions after the relinking step differ only in the position of two nodes. Hence, a number of parameters that control the process of exchanges mechanism were introduced.

2.10 Summary

Throughout this chapter we have covered the area of graph drawing including graph layout aesthetics and graph drawing techniques. Existing research in each of these fields has been explored. We have discussed several search-based techniques that were previously used in the field of graph drawing. We have also described two neighbourhood search-based techniques (tabu search and path relinking) that were not previously applied to lay out multi-criteria general graphs with straight lines, and we have highlighted their effectiveness in many applications that involve multi-criteria optimisation. In the next chapter, we describe the features of our visualisation tool along with the operations that can be performed in order to test our graph drawing algorithms and perform all the experiments conducted in this research.

Chapter 3 A Visualisation Tool

In this chapter, we describe the visualisation tool that was used to perform all the experiments discussed in this thesis. The software was implemented using Java programming language (version 1.7.0; Java HotSpot™ 64-Bit Server VM 21.0-b17 on Windows 7). It consists of two main graphical frames: a frame that includes all the operations that can be performed such as drawing graphs, loading and saving graphs, generating random graphs, and running several neighbourhood search-based graph drawing algorithms; and another frame that allows the user to control the value of the parameter of each method and the weight of each aesthetic measure. We give a detailed description of each frame in the following two sections. Note that, the code can be accessed at the Dryad Digital Repository: https://doi.org/10.5061/dryad.k082rv8.

3.1 Operations Frame

The visualisation tool allows the user to choose from a list of operations displayed in a drop-down menu inside a frame as shown in Figure 3.1. The list contains the following operations:

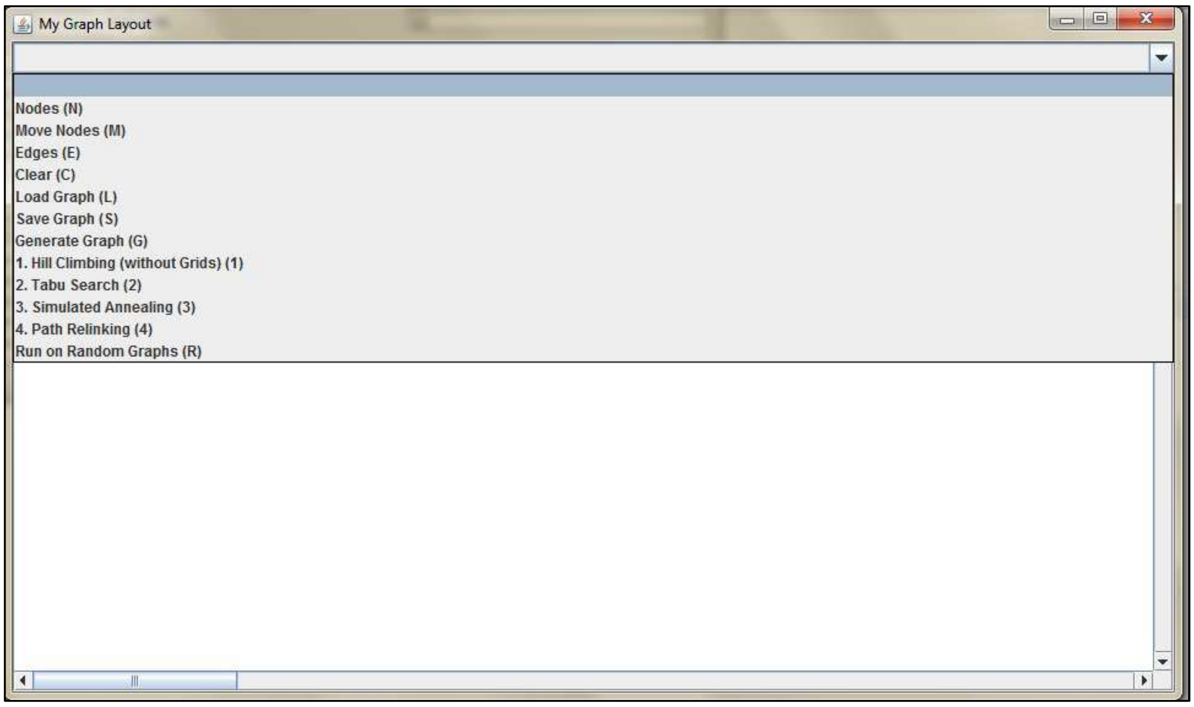


Figure 3.1 A screen shot of the drop-down menu of available operations in our visualisation tool

- ***Add nodes*** – This option allows the user to draw nodes. The user should place the mouse's cursor on the required position within the canvas and then he clicks the mouse. The node will be displayed as a small square with a side-length of 12 pixels. An automatic ID (starting from 1) will be also assigned to the drawn node (see Figure 3.2).

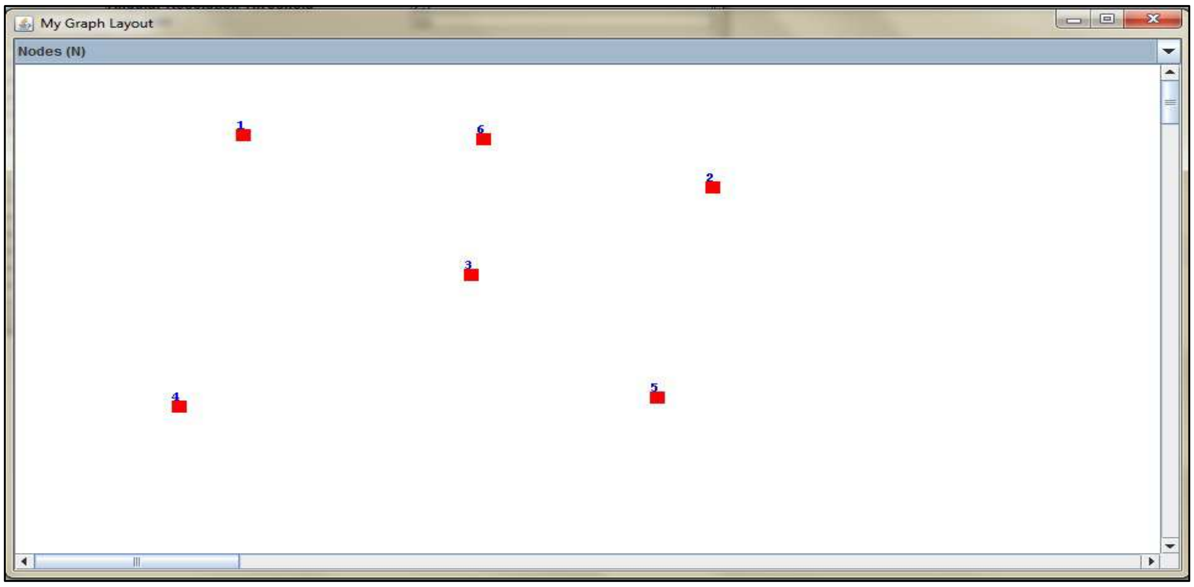


Figure 3.2 Adding nodes to the canvas

- ***Add edges*** – This option allows the user to draw edges between nodes. The user clicks the mouse over the two nodes that form the end points of the edge to be drawn (see Figure 3.3). Note that, our tool allows the user to draw simple graphs only. Self-sourcing edges and multiple edges are not allowed.

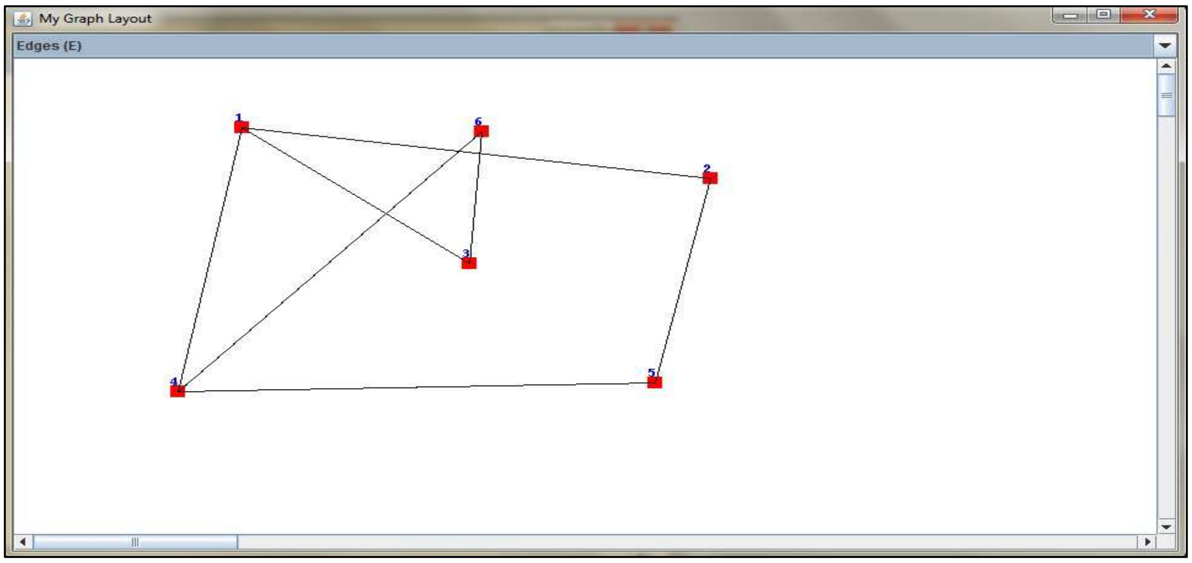


Figure 3.3 Adding edges between the nodes shown in Figure 3.2

- ***Move nodes*** **–** This option allows the user to move nodes of a graph displayed within the canvas. The user clicks the mouse over the node that he wants to move and drags it to a new position (the edge will stretch and shrink accordingly as shown in Figure 3.4). Using this option, the user can change the layout of a drawn graph. For example, the user can change the initial layout of a given graph to test its effect on the drawing algorithms.

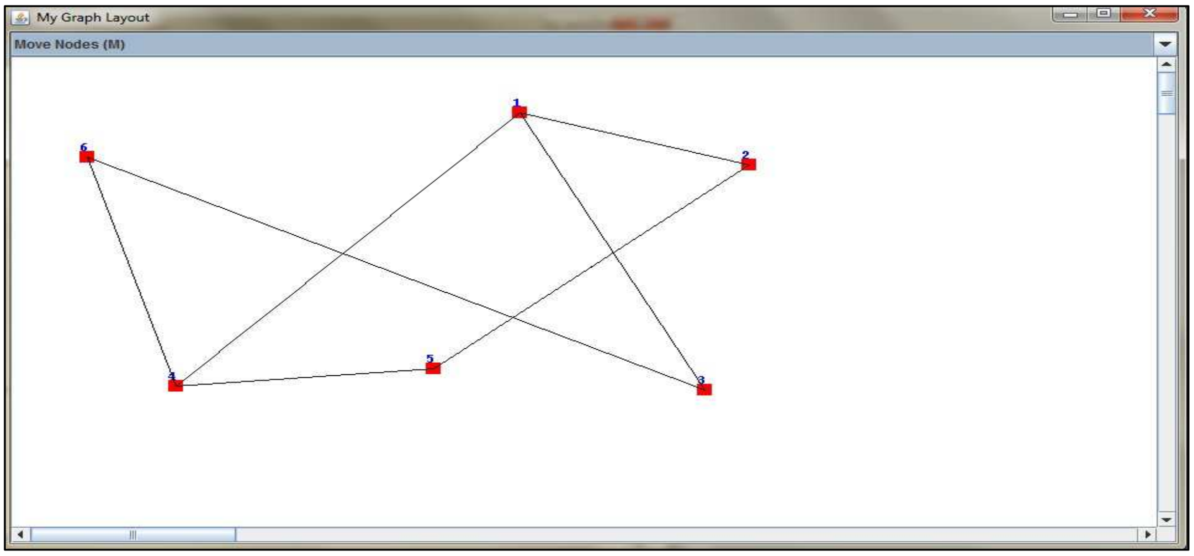


Figure 3.4 Moving nodes and stretching / shrinking edges shown in Figure 3.3

- ***Load and store graphs*** – These two options allow the user to load and save graphs from/to text files within a local directory. The file's content begins with the number of nodes in the graph followed by the coordinates of each node (i.e. the first pair of numbers represents the horizontal and vertical coordinates of the node with ID 1; the second pair of numbers represents the horizontal and vertical coordinates of the node with ID 2; and so forth). Then, the information of edges comes after. Number of nodes that are adjacent to node number 1 along with their IDs are listed first, then the same information is listed for node number 2, and so forth.
- ***Generate random graphs*** – This option allows the user to generate simple random connected graphs. The random graph generator is based on the Erdos-Renyi model (Erdos & Rényi 1960; Daudin et al. 2008). It generates randomly connected graphs. The parameters to the generator are the number of nodes (see Figure 3.5) and the density of the graph. Once the user enters the number of nodes, the tool will calculate the minimum density (i.e. minimum number of edges required to keep the graph connected which equals to number of nodes minus one) and will show it to the user (see Figure 3.6). Note that, if the user enters a value larger than the maximum density, the tool will consider the graph as a complete graph (i.e. there is an edge between every pair of nodes). Random locations for the nodes are generated based on the size of the canvas where the graph is displayed. Then, the generator chooses random nodes as the end points of the edges. All random values were generated using the random method in Java. Self-sourcing edges and multiple edges between the same pair of nodes are not allowed. Finally, the graph generator tests the connectivity of the generated graph. Only connected graphs are accepted. A sample of a randomly generated graph with 4 nodes and 4 edges (i.e. density = 0.67) is shown in Figure 3.7.

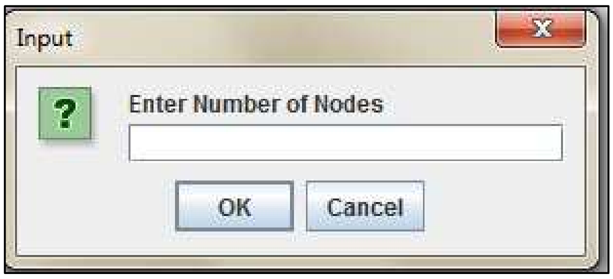
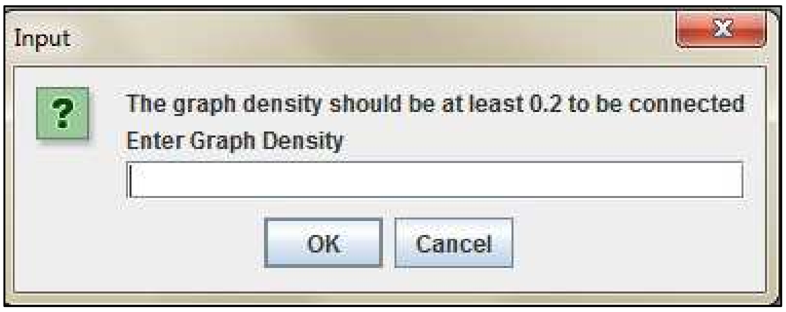


Figure 3.5 A frame prompting the user to enter number of nodes required in the random graph layout



The graph density should be at least 0.2 to be connected
Enter Graph Density

Figure 3.6 A frame prompting the user to enter the required density (showing the minimum value that can be entered)

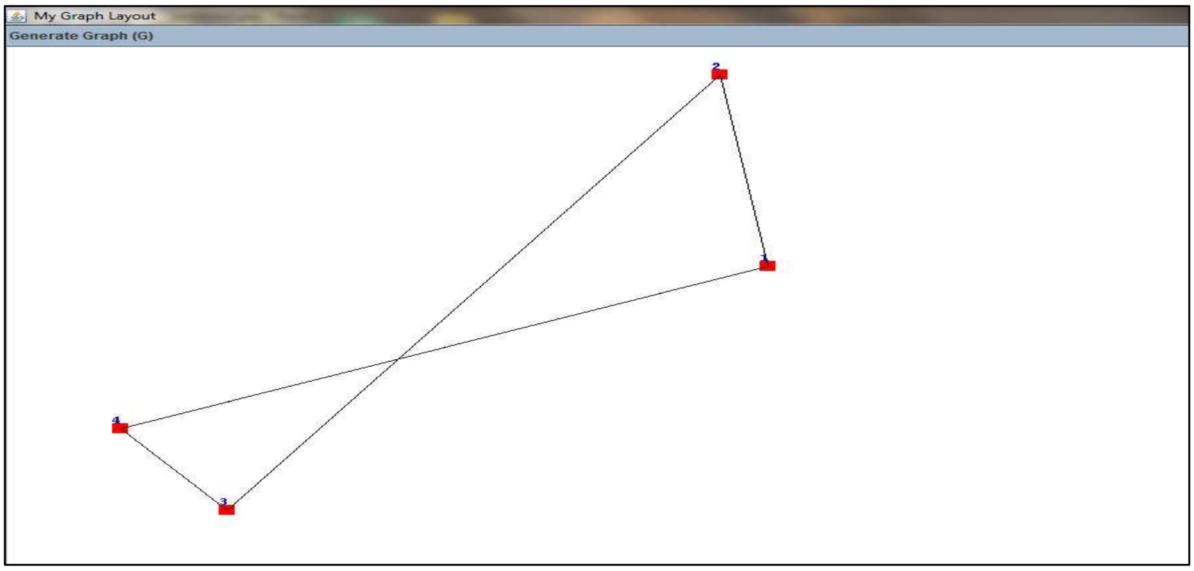


Figure 3.7 A randomly generated graph layout

In our implementation, once the random graph is displayed on the canvas, the user has the option to change the layout to another random layout by clicking on the canvas (see Figure 3.8).

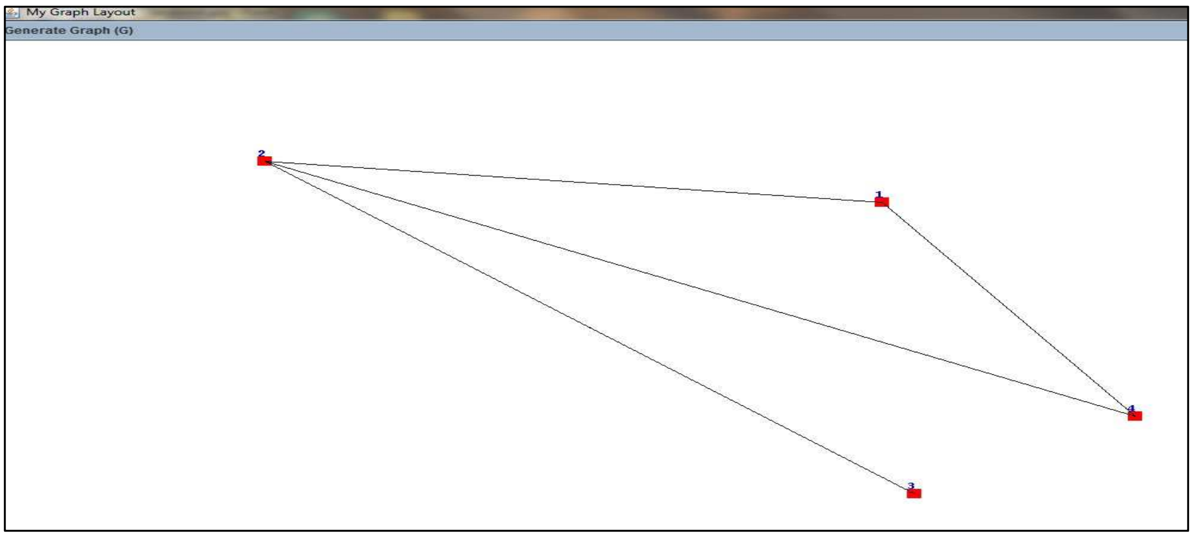
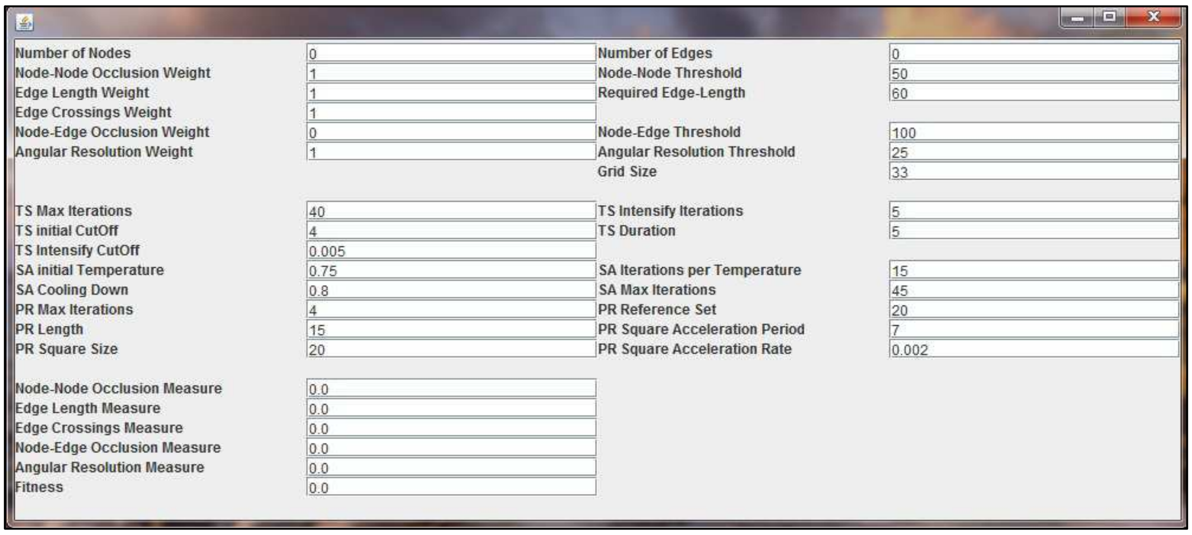


Figure 3.8 A different layout of the graph shown in Figure 3.7

- ***Select a graph drawing algorithm*** – The user has the option to select a drawing algorithm from a list of four neighbourhood search-based graph drawing algorithms: hill climbing, simulated annealing, tabu search, and path relinking (coupled with tabu search). The new layout is displayed on the canvas after applying the selected drawing algorithm. The values of parameters of each drawing algorithm can be controlled by the user using the other frame discussed in Section 3.2.
- ***Run on multiple graphs*** – This option allows the user to run a drawing algorithm on a file which contains information of multiple graph layouts. Then, it generates an output file that includes information about the fitness value of the drawn layout, the number of evaluated solutions and the execution time (in seconds) of the drawing algorithm. This operation was used in most the experiments discussed throughout this thesis. This option currently works for one drawing algorithm at a time. In order to switch to another drawing algorithm, it requires few lines of code to edit. We are planning to offer the user an easier way of algorithm's selection in the future.

3.2 Parameters and Aesthetic Measures Frame

In multi-criteria graph drawing, the weight of each metric could change for each layout as it depends on the metric in which the user prefers to focus on. Therefore, we facilitate the parameter tuning process for each method and the selection of weight for each aesthetic metric by providing another frame that contains text fields where these values can be controlled by the user (see Figure 3.9). The frame also shows the value of each individual aesthetic measure after optimisation, in addition to the value of the weighted sum of the fitness function. Additional information, such as number of nodes and number of edges of the graph displayed within the canvas, are also provided inside the frame. Note that, the number of evaluated solutions and the execution time (in seconds) of the drawing algorithm are displayed inside an alert box once the algorithm finishes execution. We will try to add these two values within this frame in the future.



Number of Nodes	0	Number of Edges	0
Node-Node Occlusion Weight	1	Node-Node Threshold	50
Edge Length Weight	1	Required Edge-Length	60
Edge Crossings Weight	1		
Node-Edge Occlusion Weight	0	Node-Edge Threshold	100
Angular Resolution Weight	1	Angular Resolution Threshold	25
		Grid Size	33
TS Max Iterations	40	TS Intensify Iterations	5
TS initial CutOff	4	TS Duration	5
TS Intensify CutOff	0.005		
SA initial Temperature	0.75	SA Iterations per Temperature	15
SA Cooling Down	0.8	SA Max Iterations	45
PR Max Iterations	4	PR Reference Set	20
PR Length	15	PR Square Acceleration Period	7
PR Square Size	20	PR Square Acceleration Rate	0.002
Node-Node Occlusion Measure	0.0		
Edge Length Measure	0.0		
Edge Crossings Measure	0.0		
Node-Edge Occlusion Measure	0.0		
Angular Resolution Measure	0.0		
Fitness	0.0		

Figure 3.9 A screen shot of the frame which allows the user to control the value of the parameter of each method and the weight of each measure

3.3 Summary

In this chapter, we described the operations and the features of our visualisation tool that we used to perform all the experiments discussed in this thesis. In the next chapter, we introduce our proposed tabu search-based technique and we demonstrate how we apply it to draw graph layouts. Then, we compare it to the most popular neighbourhood search-based algorithms: hill climbing and simulated annealing.

Chapter 4 Neighbourhood Search-based Graph Drawing including Our Proposed Tabu Search Algorithm

As discussed earlier in Chapter 2 (Section 2.5.2), there are several multi-criteria methods for graph drawing that are based on explicit cost functions that combine several metrics of graph layout quality. This approach has the advantage of allowing explicit, tuneable combinations of metrics to meet user preferences. However, such methods work slowly, typically taking a considerable time to lay out the graph. In this chapter, we want to show that we can improve the performance of such neighbourhood search-based systems by introducing the features of tabu search. This is the first time tabu methods have been applied to general graph drawing.

The main goal in this chapter is to improve the efficiency of neighbourhood search-based graph drawing algorithms by speeding up the drawing process using tabu search without sacrificing the layout quality. We are not looking for the global optimum solution, but aim to obtain a good solution quickly. Our contribution is to propose a tabu search-based approach as described in Section 4.5. But, in order to prove the efficiency of our method and its competence in relation to other neighbourhood search methods, a comparison was made with hill climbing and simulated annealing. Therefore, we introduce the implementation and parameter tuning of those two approaches first in this chapter using similar algorithms applied in Stott et al. (2011) for hill climbing and in Davidson & Harel (1996) for simulated annealing. In addition to the fact that these two methods are the most popular neighbourhood search-based methods, we chose these two methods because hill climbing is considered as one of the fastest search-based techniques to reach equilibrium, whereas simulated annealing allows more extensive search for the optimal solution and consequently usually produces better solutions compared to hill climbing (Talbi & Muntean 1993). Moreover, our tabu search method is close in concept to these methods as they share a large amount of code. In fact, the basic tabu search can be seen as simply the combination of hill climbing with short-term memories (Glover 1986). This means that it is more likely to be a fair comparison, with a low amount of bias in terms of implementation efficiency.

In this chapter, we describe the different search-based approaches which we applied in order to draw general graphs with straight lines. This is achieved by implementing neighbourhood search-based methods which draw general graphs with multiple aesthetic criteria that are used in a weighted sum fitness function to measure the quality of the graph layout. The smaller the value of the fitness function, the better the quality of the graph layout. Whilst there have been empirical studies of what may be the most effective layout criteria (Purchase 2002), we are not overly concerned with the particular criteria or their weights. Increasing the value of the weight of a metric for a certain aesthetic means that we want to show the importance of that quality measure against the other aesthetics and expecting it to be visualised in the generated layout, while the opposite is the case when the value of the weight is decreased (Davidson & Harel 1996). In our experiment, the values of the weights in the fitness function have been fixed and are the same in all approaches. With reference to the time complexity analysis performed in Davidson & Harel (1996), increasing or decreasing the value of a weight for a certain metric does not have an effect on the number of evaluated solutions performed by the algorithm.

Our fitness function follows a standard approach for search-based graph drawing methods. It is similar to the fitness function used in Davidson & Harel (1996) with some changes in the selected aesthetics. We used four metrics for measuring the quality of the graph. These metrics represent the aesthetics of: distributing nodes evenly, making uniform edge lengths, minimising edge crossings, and improving angular resolution (refer to Chapter 2 for a detailed description of each criterion). All these metrics contribute in the graph quality fitness function that is computed as follows:

$$fitness = w_1*m_1 + w_2*m_2 + w_3*m_3 + w_4*m_4$$

where w_i and m_i are the weight and the measure for criterion i respectively. The problem in a multi-criteria optimisation function is that the value of a specific measure may dominate the others. Therefore, we applied a normalisation process to ensure that the value of each measure is between 0 and 1. It is not possible to determine unified weights that work well for all types of graphs, and indeed weights can vary according to application area and user preferences. Hence, we assigned the value 1 to all the weights such that $w_1=w_2=w_3=w_4=1$.

The rest of this chapter is organised as follows: Section 4.1 demonstrates the normalisation process we applied on the criteria (metrics) used in our fitness function; Section 4.2 describes the local search space used by the three algorithms and the general procedure used for tuning their parameters; Section 4.3 describes the pseudo code for the hill climbing graph drawing algorithm along with the process of tuning its parameters; Section 4.4 describes the pseudo code for the simulated annealing graph drawing algorithm along with the process of tuning its parameters; Section 4.5 describes our proposed tabu search-based graph drawing algorithm along with the process of tuning its parameters; and Section 4.6 summarises the contents of this chapter.

4.1 Normalisation of Metrics

Multi-criteria optimisation algorithms seek to find a single optimised solution based on the weighted sum of all criteria. If all metrics get better or worse together, this conventional approach can effectively find the optimal solution. However, if there are conflicts between the metrics, then there is no single optimal solution. In most cases, there are infinitely many optimal solutions. An optimal solution in the multi-criteria optimisation context is a solution where there is no other feasible solution that improves the value of at least one criterion without deteriorating any other criterion. This is the notion of Pareto Optimality (Sunar & Kahraman 2001; Kim & de Weck 2005).

The weighted sum formula allows the multi-criteria optimisation problem to be transformed into a single criterion optimisation function that is constructed as a sum of objective functions (metrics) m_i multiplied by weighting coefficients w_i (Grodzevich & Romanko 2006). The problem is formulated as follows:

$$\min \sum_{i=1}^{k} w_i m_i(G)$$

such that G is a set of nodes and edges that form a graph, where $w_i \geq 0, \forall i = 1, \dots, k$.

The problem in the multi-criteria optimisation function is that the value of a single measure might largely dominate the others. Also, as different measures can have different magnitudes, the normalisation of measures is required in order to obtain a solution consistent with the weights assigned by the decision-maker who has insights into the problem and is able to express relative importance of the measures.

In our graph layout problem, we have four different metrics that contribute in a single weighted sum optimisation function. Each measure has a different scale of values (i.e. the range of values of each measure differs from one measure to another). Furthermore, the node-node occlusion measure (as described in Chapter 2 Section 2.4) might have a maximum value of infinity (when two nodes have the same coordinates). Therefore, normalising the values to a unified range (i.e. a range between 0 and 1) is required.

We normalised the values of measures using the min-max method (Kotsiantis et al. 2006; Shalabi et al. 2006). This method assumes that the minimum (F_{min}) and the maximum (F_{max}) values of a measure (m) are known. Then it uses the following function for normalisation:

$$\frac{m - F_{min}}{F_{max} - F_{min}}$$

This formula was directly applied on the measure of edge crossings since the minimum and maximum possible values for edge crossings can be easily calculated as follows:

$$F_{min} = 0,$$

$$F_{max} = E * (E - 1) / 2 \text{ , where } E \text{ is the number of edges.}$$

However, the normalisation process was slightly different with the measures of node-node occlusion, edge lengths, and angular resolution as the calculation of maximum value of these measures is not straight forward and in some cases it could reach infinity. Therefore, we performed the following process to normalise those measures:

i. As the graph drawing algorithm goes through several iterations searching for candidate solutions, in the first iteration, we compute the value of each measure and we consider that solution as an initial solution vector.

ii. In all the subsequent iterations, for each measure, we compute the current maximum and minimum values of all the generated values (tracking a history of values) for each measure in order to use in the calculation of a normalised value between 0 and 1. For example, the normalised value of measure m at iteration i (m_i),

$$NormValue_i = \frac{m_i - Min_i}{Max_i - Min_i}$$

where Min_i and Max_i are the minimum and the maximum values of the measure at the i^{th} iteration.

iii. This process is performed at each iteration until a solution is found.

This is considered as an estimation of the normalised value for the measure. Calculating the normalised values using this method will not affect the performance of our drawing algorithm because we just compare the newly generated values with the current maximum (minimum) and we update the value accordingly.

After applying all the above calculations, the value of each measure lies between 0 and 1 and none of the measures dominates the others. Thus, the value of our fitness function is always a small non-negative value such that the maximum value is 4 and the minimum value is 0 since our fitness function consists of four measures. This normalisation process is also used in the field of neural networks to avoid neuron saturation where a neuron predominantly outputs values close to the asymptotic ends of the bounded activation function (Jayalakshmi & Santhakumaran 2011).

Our graph drawing algorithms are applied to lay out general graphs that might have different properties. Therefore, assigning weights to the measures in the weighted sum formula would be an interactive process with decision-makers (users) who have background in graph layout. We cannot determine unified weights that work properly for any graph. Thus, the weights should be assigned by decision-makers according to their preferences of which measure they want to test. In our experiments, we assign the value 1 to all weights in order to avoid the domination of a measure over another.

4.2 Common Procedures between Graph Drawing Algorithms

In this section, we describe the basic local search procedure used in the three neighbourhood search-based graph drawing algorithms discussed in this chapter. We also provide a detailed description of the parameter tuning process that we applied to tune the value of each parameter in each algorithm.

4.2.1 Local Search Space

In all the algorithms described in this work, we use a systematic exploration of the search space. For each node, we search the points (candidate solutions) around a square centred on the node at a given distance, as shown in Figure 4.1. Eight points around the square are checked (up, down, left, right, and the four corners). We compute the fitness value at each candidate solution, and we select the candidate solution that gives the lowest fitness value (`currentFitness`). In the case that there are multiple candidate solutions that share the lowest fitness value, we select the first encountered candidate solution starting from the right point around the square and move along the points of the square in a clockwise direction. This is how the fitness tie-breaks in all the methods discussed in this work.

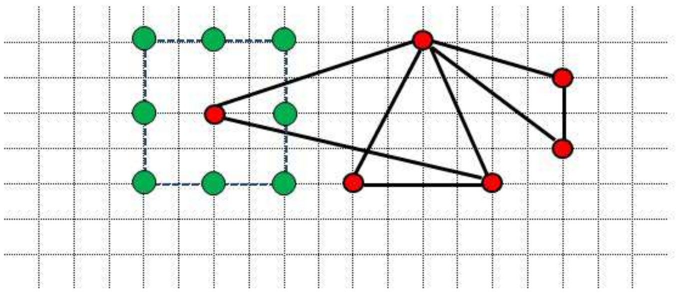


Figure 4.1 The points around the square represent the candidate solutions at each node

Note that, using a geometric shape for defining a search space in the field of graph drawing was used earlier in Davidson & Harel (1996), and Stott et al. (2011) where a circle and a rectangle had been respectively used. However, since evaluating a multi-criteria fitness value is a lengthy process, we restrict the movements to eight points only to avoid the long execution time for re-evaluating the value of the fitness function with a large number of evaluated solutions. We use the same neighbourhood searching strategy with all the methods included in this work in order to make a fair comparison. This searching strategy can be easily adjusted with our implementation by increasing the number of repetitions from eight points to any larger number, but the execution time would be significantly longer.

4.2.2 Parameter Tuning Procedure

Each method has a different number of parameters that affect the performance of the method and the quality of the layouts generated by these methods. The parameters calibration process is a key step in the development of any algorithm. Several experiments were conducted to tune the parameters of the three methods. The experiments show the effect of increasing and decreasing the values of the parameters on the method's performance and the layout's quality.

Parameter tuning is a common practice in search-based methods. Typically, one parameter is tuned at a time that may cause some suboptimal choices, since parameters often interact in a complex way. However, the simultaneous tuning of more parameters leads to an enormous amount of experiments. There are some technical drawbacks to parameter tuning based on experimentation that can be summarised as follows (Eiben et al. 1999):

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- The process of parameter tuning is time consuming, even if the parameters are optimised one by one, regardless to their interactions.
- For a given problem, the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.

However, many researchers (Davidson & Harel 1996; Rosete-Suárez et al. 1999; Pacheco & Marti 2006; Gendreau & Potvin 2014) used the following process for tuning parameters:

i. Perform exploratory tests on a wide range of values for each parameter in order to select a robust set of initial values.

ii. Perform a systematic incremental procedure for testing the values of each single parameter at a time while fixing the values of the rest of the parameters at what appears to be reasonable.

In computational experiments, it is recommended to divide the datasets into two subsets; one that is used in the algorithm design and the tuning of the parameters, whereas the other subset is used in the final experimentation after the parameters are calibrated. This is necessary for avoiding overfitting, i.e. the tuned parameters might be good for the dataset at hand, but they produce poor results in general with different datasets (Gendreau & Potvin

2014). Overfitting can be beneficial if we are trying to find the best set of values to parameters for a specific type of graphs with certain properties but it causes a problem when we are looking for more general results (Hawkins 2004).

In the next sections, we describe the basic neighbourhood search-based graph drawing algorithms for hill climbing and simulated annealing, followed by our tabu search-based approach for graph layout. For each algorithm, we provide a detailed description of the parameters that the algorithm requires along with the results of the parameter tuning process.

4.3 Hill Climbing

Hill climbing has been applied as a multi-criteria search-based method in the field of graph drawing in Rosete-Suárez et al. (1999), and Stott et al. (2011). Algorithm 4.1 shows an overview of the process for a straightforward, generic hill climbing method for graph layout.

4.3.1 Algorithm

The algorithm operates in the following manner: first, we compute the fitness value of the initial layout (`layoutFitness`). Then a local search procedure is implemented, as described in the previous section. The square size starts with an initial predefined value, (*`initialSquareSize`*). In order to intensify the searching process, the square size is reduced when none of the candidate solutions at the current square size makes an improvement to the current solution. `SmallerSquareSize()` is a function that reduces the current square size (`squareSize`) by a predefined reduction rate (*`squareReduction`*) using the following formula:

```
squareSize = squareSize / squareReduction
```

The whole process of searching is repeated as long as the square size is of a positive value. See Algorithm 4.1.

Given:

Connected Graph `G(V,E)`: `V` is a set of nodes and E ⊆ (V×V) is a set of edges.

`initialSquareSize`: predefined size of a square where candidate solutions are located on its border.

`squareReduction`: predefined value which represents the rate of reduction for the size of the square.

Algorithm:

```
allOffsets = {(1,1), (1,0), (1,-1), (0,-1), (-1,-1), (-1, 0), (-1, 1), (0, 1)}
squareSize = initialSquareSize
layout = RandomizeLayout(G) /* layout maps each node in G to an (x,y) position */
while squareSize > 0 do
  layoutFitness = Fitness(layout)
  for v in V do
    currentPos = layout[v] /* position currently associated with node v */
    currentFitness = Fitness(layout)
    for scaledOffset in {(squareSize*x, squareSize*y) | (x,y) in allOffsets}
      candidatePos = currentPos + scaledOffset /* vector addition */
      if (Fitness(candidatePos) < currentFitness)
        layout[v] = candidatePos
        currentFitness = Fitness(layout)
      end if
    end for
  end for
  if (currentFitness >= layoutFitness) /* in case of no improvement in layout fitness*/
    squareSize = SmallerSquareSize(squareSize, squareReduction)
  end if
end while
```

Algorithm 4.1 Hill climbing graph drawing algorithm

4.3.2 Parameter Tuning

The hill climbing algorithm is affected by two parameters: the initial value of the square size used to determine the neighbourhood solutions (*initialSquareSize*) and the value used to reduce the size of the square (*squareReduction*).

In order to tune the parameters of this algorithm and the other algorithms in this research, several experiments were conducted to calibrate the parameters of each method. We performed exploratory tests on a wide range of values for each parameter in order to select a robust set of initial values. Then we ran a systematic incremental procedure for each single parameter at a time while fixing the values of the other parameters. This is similar to the tests conducted in Davidson & Harel (1996), Rosete-Suárez et al. (1999), Pacheco & Marti (2006), and Gendreau

& Potvin (2014). Since Erdos-Renyi graphs with the same parameters are known to possess very similar characteristics (Bollobás 1998; Titiloye & Crispin 2012), we generated 100 random connected graphs based on Erdos-Renyi model that were divided into five sets such that the graphs in each set had a different number of nodes and edges compared to the graphs in the other sets. Hence, each set consisted of 20 test cases with the same number of nodes and edges but with different initial layouts. The characteristics of the five sets are described in Table 4.1. Since all our experiments are applied on undirected simple graphs, we use the following formula for computing the density of a graph (Coleman & Moré 1983):

$$Density = \frac{2|E|}{|V|(|V|-1)}$$

Table 4.1 The characteristics of graph datasets used in parameter tuning for the hill climbing algorithm

Graph Set	Nodes	Edges	Density	Label
1	50	153	0.125	N50E153
2	100	544	0.110	N100E544
3	150	1173	0.105	N150E1173
4	200	1890	0.095	N200E1890
5	250	2645	0.085	N250E2645

The parameters' tuning process has passed through two phases. In the first phase we try to find a proper set of values of parameters that gives the smallest fitness (best quality), whereas in the second phase we try to find a set of values that gives the smallest number of evaluated solutions.

i. Phase I

In phase I, we tested the hill climbing drawing algorithm on the 100 test cases for four different values of *initialSquareSize*: 64, 128, 256, 512, and four different values of *squareReduction*: 2, 4, 6, 8. We tested all combinations of these values in an attempt to obtain the parameters' values that give the best graph layout quality among all possible combinations. We started the process by fixing the value of *squareReduction* to 2 and changing the values of *initialSquareSize* according to the list of values mentioned above. We applied the same process for all the values of *squareReduction*. Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.5 show the values of the fitness function generated by the hill climbing drawing algorithm when we use all combinations of the parameters' values listed above. The figures show that the value of the fitness function decreases when the value of *initialSquareSize* increases. In this phase of testing, we looked for the combination of parameters' values that give the smallest fitness value (best quality) compared to all other combinations regardless of the number of evaluated solutions performed by the algorithm. The best values we got in this phase were 512 for *initialSquareSize* and 2 for *squareReduction*.

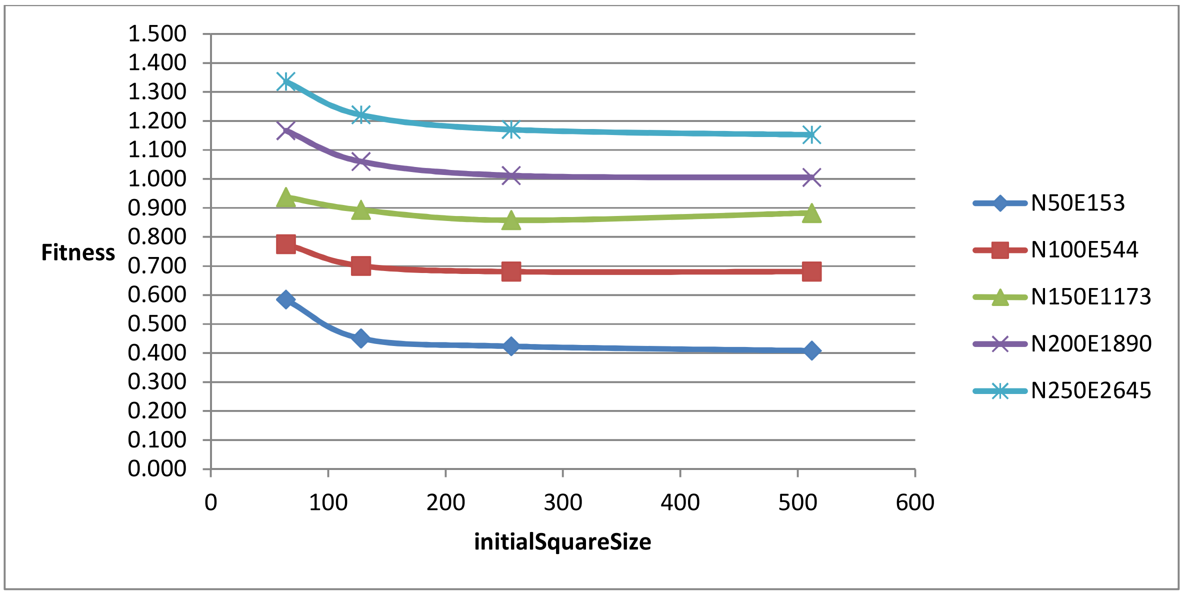


Figure 4.2 Hill Climbing - Fitness value when squareReduction = 2 (phase I)

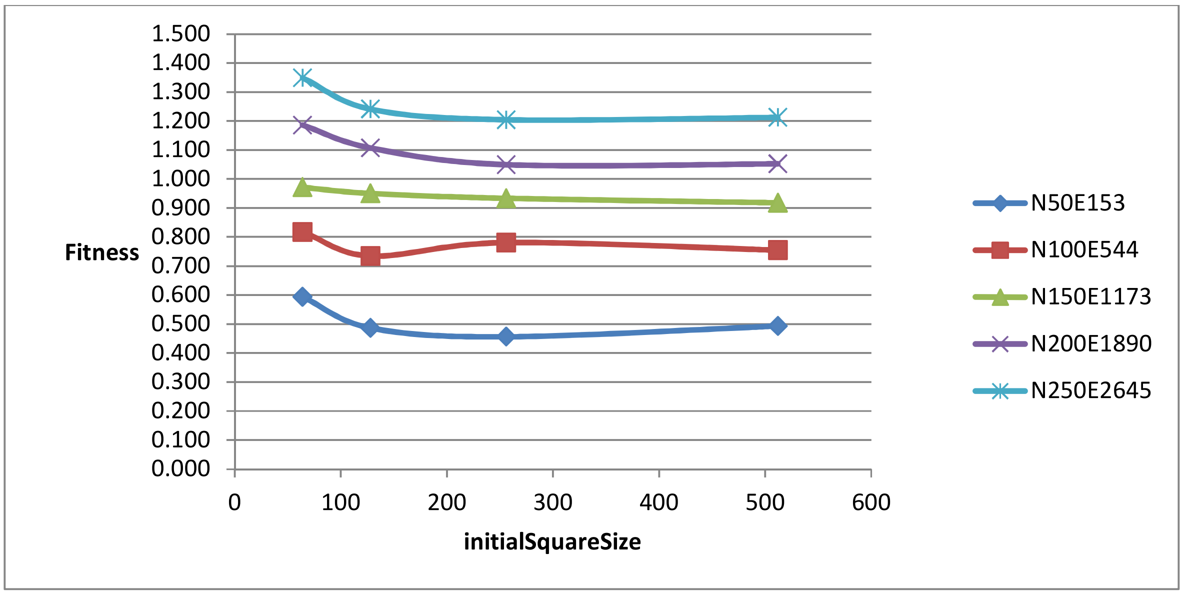


Figure 4.3 Hill Climbing - Fitness value when squareReduction = 4 (phase I)

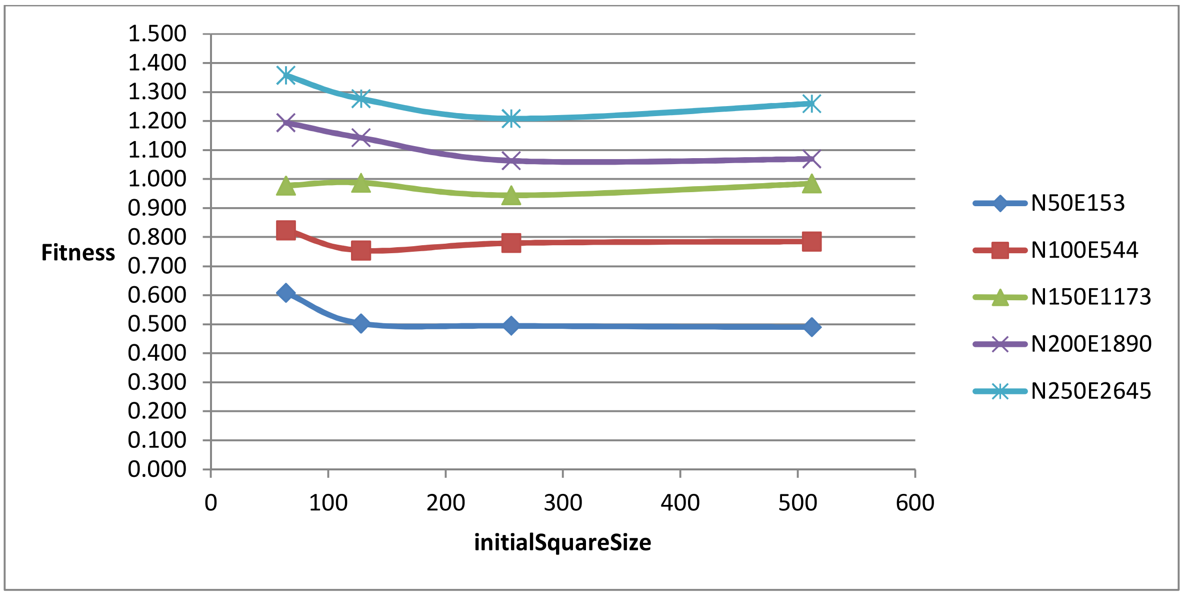


Figure 4.4 Hill Climbing - Fitness value when squareReduction = 6 (phase I)

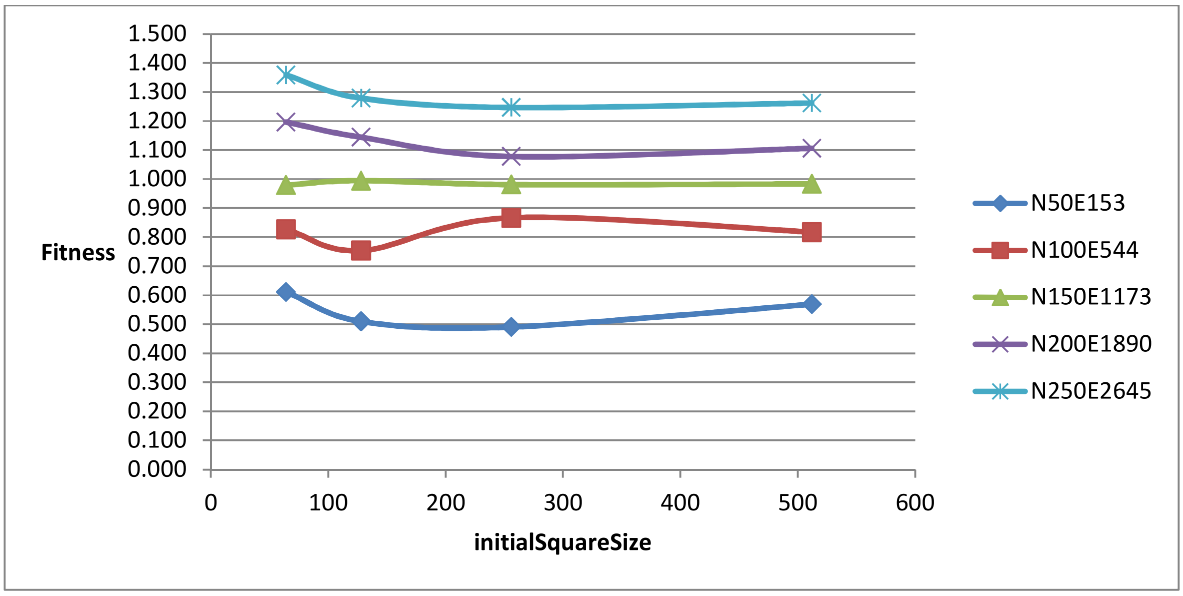


Figure 4.5 Hill Climbing - Fitness value when squareReduction = 8 (phase I)

ii. Phase II

In phase II of parameter tuning, we focus on the performance of the algorithm (i.e. number of evaluated solutions). The target is speeding up the process of drawing a good graph layout but not necessarily the best layout. To do so, we took a view that a good-enough graph layout is a layout in which its fitness value is slightly greater than the best fitness value produced in the experiments of phase I. Therefore, we took the values of the fitness function produced by the selected parameters' values in phase I and we increased them by 12.5%. Then we ran the hill climbing drawing algorithm until it reached equal fitness values to the target fitness values or no further improvement in the fitness value could be made. Afterwards, we picked the most appropriate parameters' values that gave a good layout with a small number of evaluated solutions.

In this experiment, we tested once more the following values for *initialSquareSize*: 64, 128, 256, 512 and the values 2, 4, 6, 8 for *squareReduction.* We followed the same process performed in phase I by fixing the value of *squareReduction* to 2 and changing the values of *initialSquareSize* according to the list of values given above. Then we repeat this for all the values of *squareReduction*. The tables from Table 4.2 to Table 4.5 and the figures from Figure 4.6 to Figure 4.9 show the fitness function values and number of evaluated solutions generated by the hill climbing algorithm in phase II.

According to these tables and figures, we recognised that the values of the parameters that gave small fitness values (good quality) with a small number of evaluated solutions were: 512 for *initialSquareSize* and 4 for *squareReduction.* Using these two values for the parameters made the hill climbing algorithm produce close fitness values to the target fitness values with a limited number of evaluated solutions compared to the other parameters' values. We could have used the value 2 for *squareReduction* since it produced graph layouts with fitness values that were slightly better than the graph layouts produced by the algorithm when *squareReduction* equals to the value 4. However, the latter generated a lower number of evaluated solutions compared to the former value, and since there is only a slight difference between the values of the fitness function produced using these two values, we selected the value 4 for *squareReduction.*

Table 4.2 Hill Climbing - Fitness value when squareReduction = 2 (phase II)

	Fitness				
initialSquareSize	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
64	0.591	0.821	0.996	1.168	1.336
128	0.487	0.766	0.991	1.127	1.290
256	0.474	0.764	0.981	1.125	1.291
512	0.453	0.760	0.985	1.123	1.288

Target	0.408	0.681	0.883	1.006	1.152

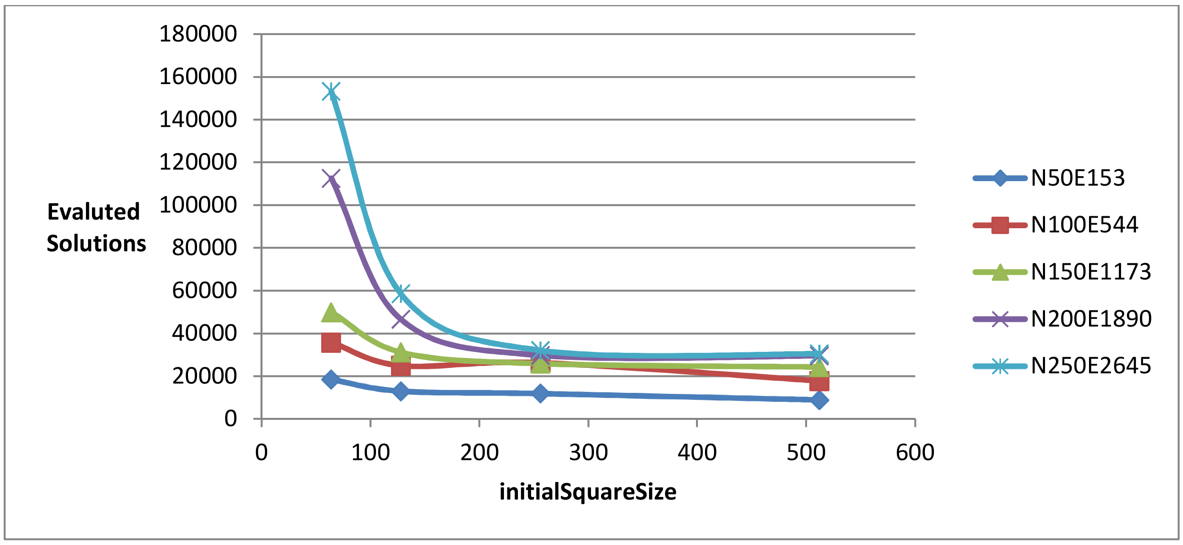


Figure 4.6 Hill Climbing -Number of evaluated solutions when squareReduction = 2 (phase II)

Table 4.3 Hill Climbing - Fitness value when squareReduction = 4 (phase II)

	Fitness				
initialSquareSize	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
64	0.599	0.855	1.019	1.187	1.348
128	0.505	0.784	1.017	1.145	1.288
256	0.487	0.812	1.013	1.129	1.299
512	0.500	0.800	0.996	1.120	1.297

Target	0.408	0.681	0.883	1.006	1.152

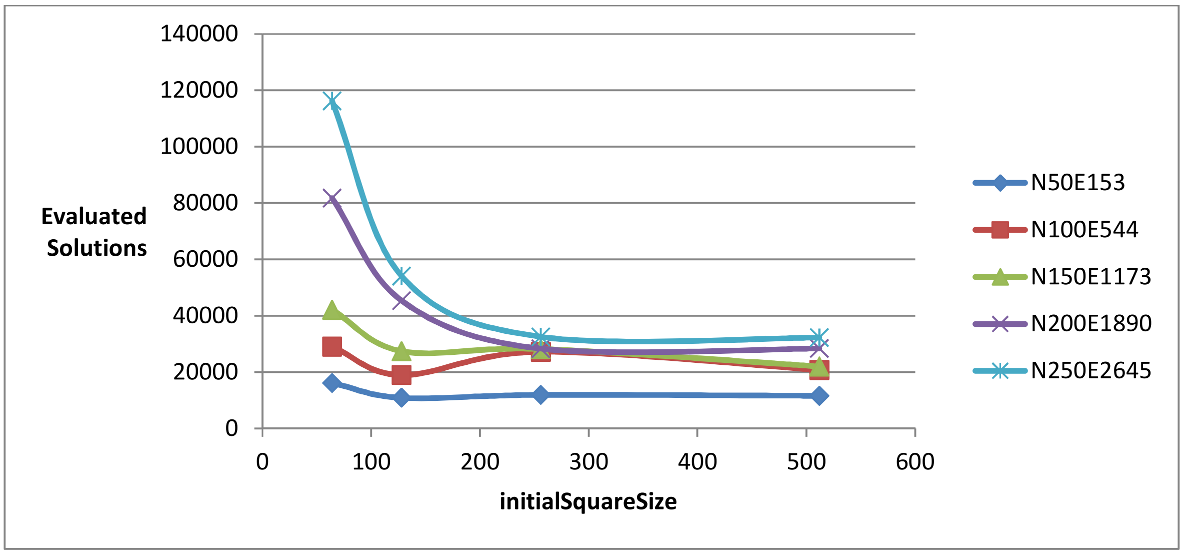


Figure 4.7 Hill Climbing - Number of evaluated solutions when squareReduction = 4 (phase II)

Table 4.4 Hill Climbing - Fitness value when squareReduction = 6 (phase II)

	Fitness				
initialSquareSize	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
64	0.612	0.860	1.021	1.194	1.357
128	0.518	0.792	1.033	1.175	1.314
256	0.506	0.818	1.017	1.126	1.285
512	0.509	0.821	1.050	1.124	1.317

Target	0.408	0.681	0.883	1.006	1.152

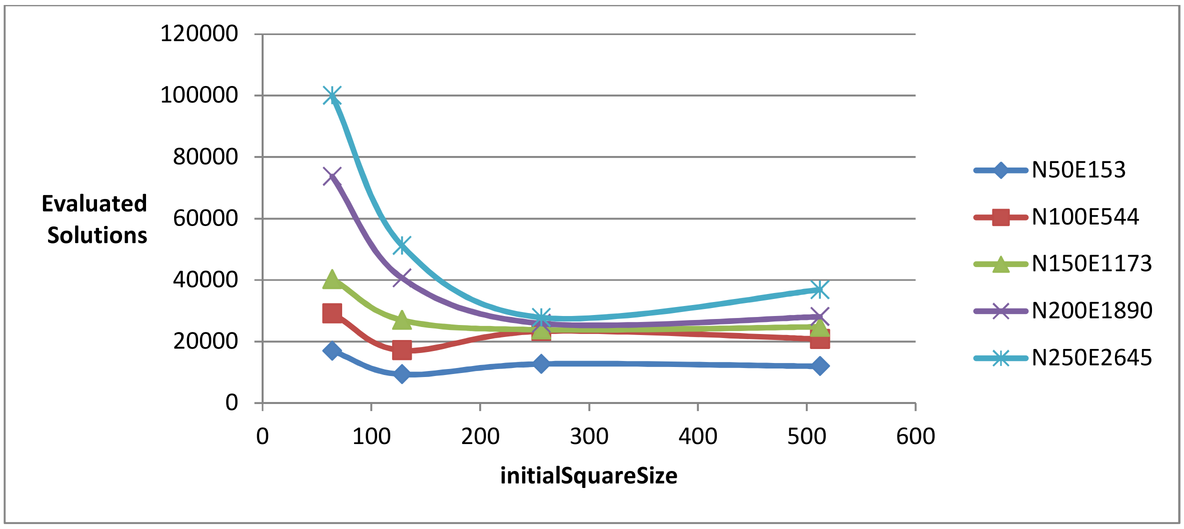


Figure 4.8 Hill Climbing - Number of evaluated solutions when squareReduction = 6 (phase II)

Table 4.5 Hill Climbing - Fitness value when squareReduction = 8 (phase II)

	Fitness				
initialSquareSize	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
64	0.615	0.861	1.022	1.197	1.358
128	0.521	0.792	1.038	1.174	1.314
256	0.509	0.881	1.039	1.128	1.307
512	0.571	0.831	1.022	1.134	1.303

Target	0.408	0.681	0.883	1.006	1.152

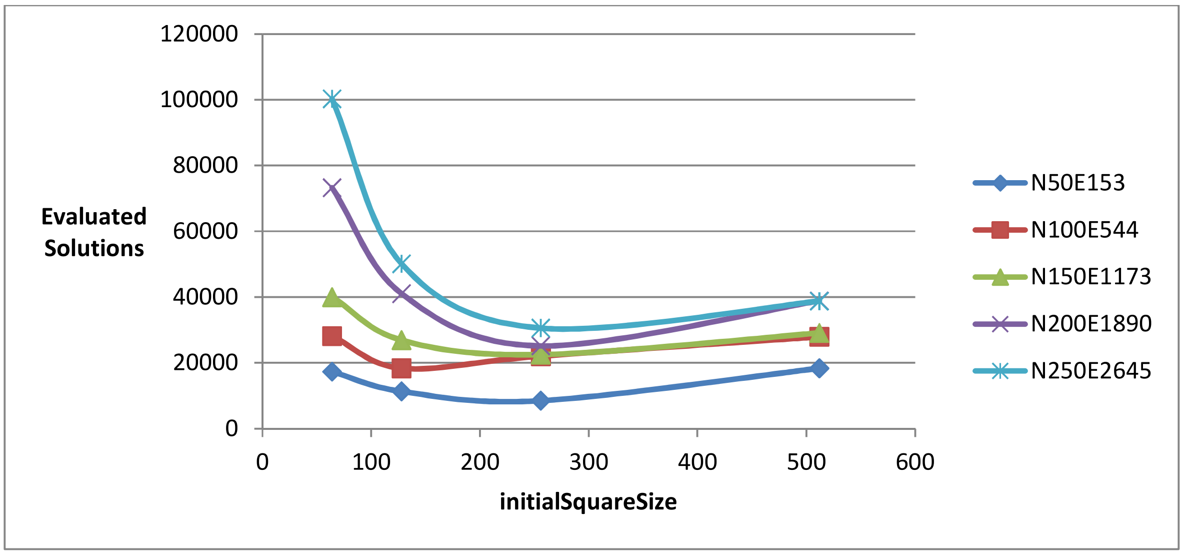


Figure 4.9 Hill Climbing - Number of evaluated solutions when squareReduction = 8 (phase II)

4.4 Simulated Annealing

Simulated annealing was first used for the graph layout problem in Davidson & Harel (1996). It has been used to draw general undirected graphs with straight edges taking into account several drawing aesthetics. An overview of a generic implementation for simulated annealing used in drawing graph layouts is shown in Algorithm 4.2.

4.4.1 Algorithm

The algorithm starts by choosing an initial graph layout and an initial temperature (*`initialTemp`*). Then it repeats the following steps for fixed number of iterations (*`maxIterations`*): a new candidate solution is chosen from the neighbourhood of the current solution using the same neighbourhood solution selection process described in section 4.2.1 but the selection is performed at random (i.e. moving only one node in the current layout to a new location on the points of the surrounding square to get a new layout). The fitness value of the new candidate solution is computed and compared to the fitness value of the current solution. The candidate solution becomes the new current solution if the fitness value of the candidate solution is less than the fitness value of the current solution. Also, there is a probability of selecting the candidate solution as the new current solution even if its fitness value is larger than the fitness value of the current solution. This happens if the difference between the fitness values satisfies the following condition:

$$e^{-(candidateFitness - currentFitness) / t} \leq \text{random}[0,1)$$

where *t* is the current temperature of the system.

As the general simulated annealing algorithm dictates, a series of moves is attempted at each temperature (i.e. the annealing process keeps searching for candidate solutions using the same temperature for a certain number of iterations). Therefore, we have to decide when to change the temperature and how to change it. `iterPerTemp` is the predefined value that represents the number of iterations needed to search for candidate solutions at each temperature.

The cooling down schedule is one of the most crucial parts of the annealing algorithm. As we start with an initial temperature (`initialTemp`), the temperature should be decreased after a predefined number of iterations (`iterPerTemp`). We follow most researchers (Davidson & Harel 1996) in applying the following rule as referenced in Algorithm 4.2 by the `CoolingDown()` function:

$$t_{new} = t_{old} * \mathtt{coolDown}$$

where *t* represents the temperature and `coolDown` is a predefined value that represents the temperature reduction rate. Slow cooling may improve the results but at a cost of increasing running time. In addition to cooling down the temperature, the size of the square, in which the candidate solutions of the current solution lie, should also be reduced. `SmallerSquareSize()` is the function we used to reduce the size of the square as in hill climbing. However, our experiments showed that a slower reduction rate would give better graph layouts. According to Davidson & Harel (1996), we used the following formula:

`squareSize = squareSize – (`*`initialSquareSize / squareReduction)`*

Given:

Connected Graph `G(V,E)`: V is a set of nodes and E ⊆ (V×V) is a set of edges.

`initialSquareSize`: predefined size of a square where candidate solutions are located on its border.

`squareReduction`: predefined value which represents the rate of reduction for the size of the square.

`maxIterations`: predefined value for the number of iterations for running the drawer.

`iterPerTemp`: predefined value for the required number of iterations at each temperature.

`initialTemp`: initial temperature used in the annealing process.

`coolDown`: predefined value for the temperature cooling down rate.

Algorithm:

```
allOffsets = {(1,1), (1,0), (1,-1), (0,-1), (-1,-1), (-1, 0), (-1, 1), (0, 1)}
squareSize = initialSquareSize
layout = RandomizeLayout(G) /* layout maps each node in G to an (x,y) position */
t = initialTemp
iteration = 0
while iteration < maxIterations do
  for i:= 1 to iterPerTemp do  /* number of iterations at each temperature */
    for v in V do
      currentPos = layout[v] /* position currently associated with node v */
      currentFitness = Fitness(layout)
      generate random scaledOffset in {(squareSize*x,squareSize*y) | (x,y) in allOffsets}
      candidatePos = currentPos + scaledOffset    /* vector addition */
      if (Fitness(candidatePos) < currentFitness)
        layout[v] = candidatePos
        currentFitness = Fitness(layout)
      else
        costDiff = Fitness(candidatePos) - currentFitness
        if ($e^{-costDiff / t}$ < random[0,1))
          layout[v] = candidatePos
          currentFitness = Fitness(layout)
        end if
      end if
    end for
  end for
  t = CoolingDown(t, coolDown)
  squareSize = SmallerSquareSize(squareSize, squareReduction)
  iteration = iteration + 1
end while
```

Algorithm 4.2 Simulated annealing graph drawing algorithm

4.4.2 Parameter Tuning

The performance of the simulated annealing drawing algorithm is influenced by four parameters: the number of iterations for running the algorithm (*maxIterations*), the number of iterations at each temperature (*iterPerTemp*), the initial temperature used in the annealing process (*initialTemp*), and the temperature cooling down factor (*coolDown*).

Simulated annealing is characterised as a slow search-based method. It is also a stochastic method unlike hill climbing and tabu search. Thus, in order to speed up the testing process, the process for generating the graphs used for testing was a bit different than the one used in the previous method. We generated 10 random connected graphs, based on Erdos-Renyi model, that were divided into five sets (as described previously in Table 4.1) such that each set had two graphs with different initial layouts. Then, for each graph in each data set, we run the simulated annealing drawing algorithm for 10 runs and we find the median of the results.

The parameters of simulated annealing are dependent. Increasing or decreasing the value of one parameter affects the values of the other parameters. Therefore, we followed an incremental testing process divided into three phases described as follows: in phase I, we started with one parameter, tested it thoroughly with different values, and selected the value which produced the best layout compared to the other values. We fixed the value of the first parameter and we moved to testing another parameter in the same manner, and so forth. In this phase, we were searching for the most appropriate values of the parameters that make the simulated annealing algorithm produce good layout regardless of the number of evaluated solutions performed by the drawer. Simulated annealing used the same neighbourhood searching technique that was used in hill climbing. In the previous phase, we used an initial square size of 256. However, after performing a complete testing on the parameters of hill climbing, an initial square size of 512 has produced graph layouts with better quality and a fewer number of evaluated solutions performed by the algorithm. Therefore, in phase II, we repeated the same testing process that we performed in phase I using the best initial square size parameter, as described in the parameter tuning process of hill climbing. In phase III, we mainly focus on choosing the parameters which speed up the algorithm's performance (i.e. number of evaluated solutions).

i. Phase I

We started the testing process with the first parameter *maxIterations* by testing it with the following values: 30, 40, 50, 60, 70, whereas the remaining parameters were set to some arbitrary values such that *iterPerTemp = 20*, *initialTemp = 0.5*, and *coolDown = 0.8*. These arbitrary values were very close to the values used in Davidson & Harel (1996). Figure 4.10 shows the effect of *maxIterations* on the fitness value. The simulated annealing drawing algorithm produced graph layouts with good fitness values when the value of *maxIterations* was 40 and 50. There is no significant difference between the two values. However, we chose the value 40 because it generates a lower number of evaluated solutions (i.e. faster).

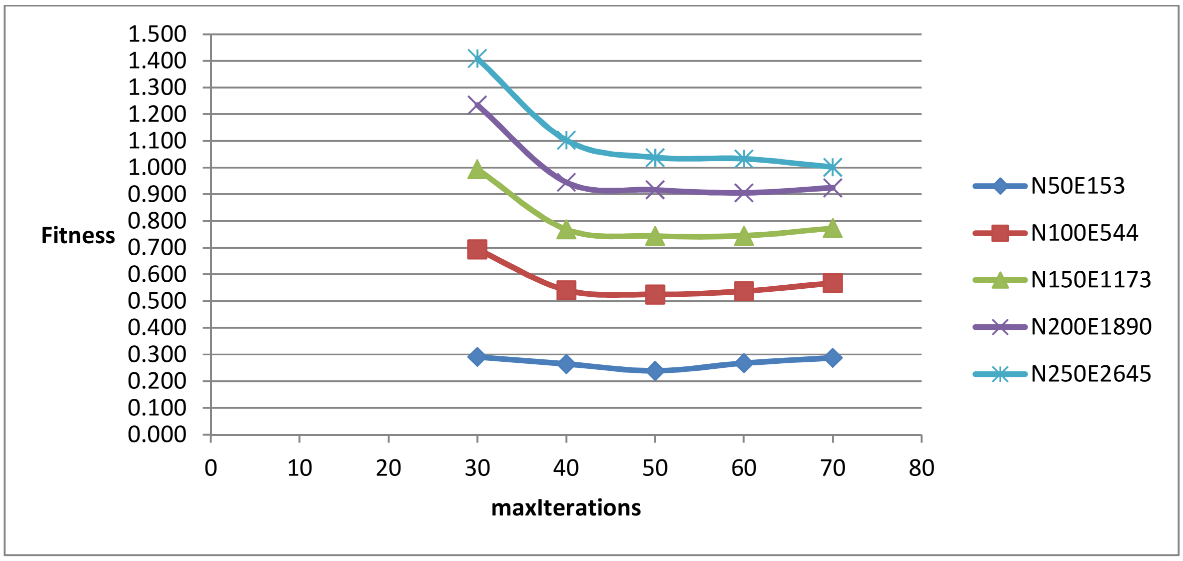


Figure 4.10 Simulated Annealing - Fitness values with the maxIterations parameter (phase I)

After setting the value of *maxIterations* to 40, we moved on to test the value of *iterPerTemp* with the values: 10, 15, 20, 25, 30, 35, 40. Figure 4.11 shows that increasing the value of this parameter produces graphs with better layouts. As shown in the figure, the fitness values were close starting from the value 25 onwards. Thus, we chose the value 25 for *iterPerTemp*.

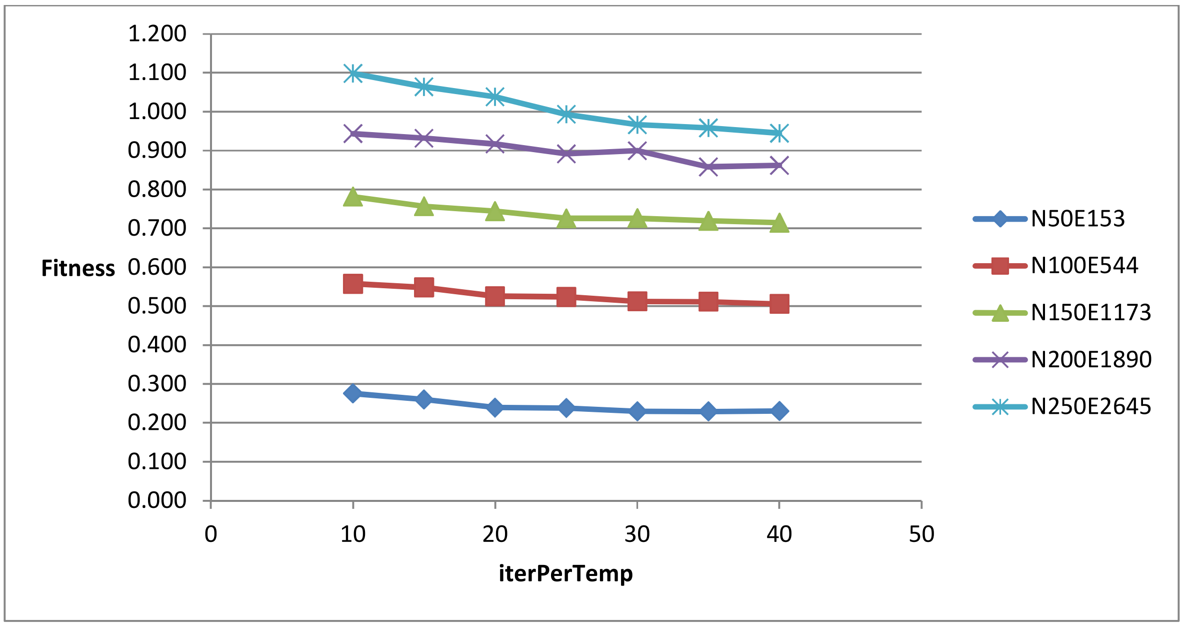


Figure 4.11 Simulated Annealing - Fitness values with the iterPerTemp parameter (phase I)

As opposed to *iterPerTemp*, increasing the value of the temperature parameter *initialTemp* produces graph layouts with poor quality. We tested the *initialTemp* parameter with the values: 0.5, 2.5, 4.5, 6.5. According to Figure 4.12 which shows the effect of the temperature on the quality of the graph layout, we chose the value 0.5 for *initialTemp* since it is the best value that produced graphs with good layouts compared to all the other values used in the testing process.

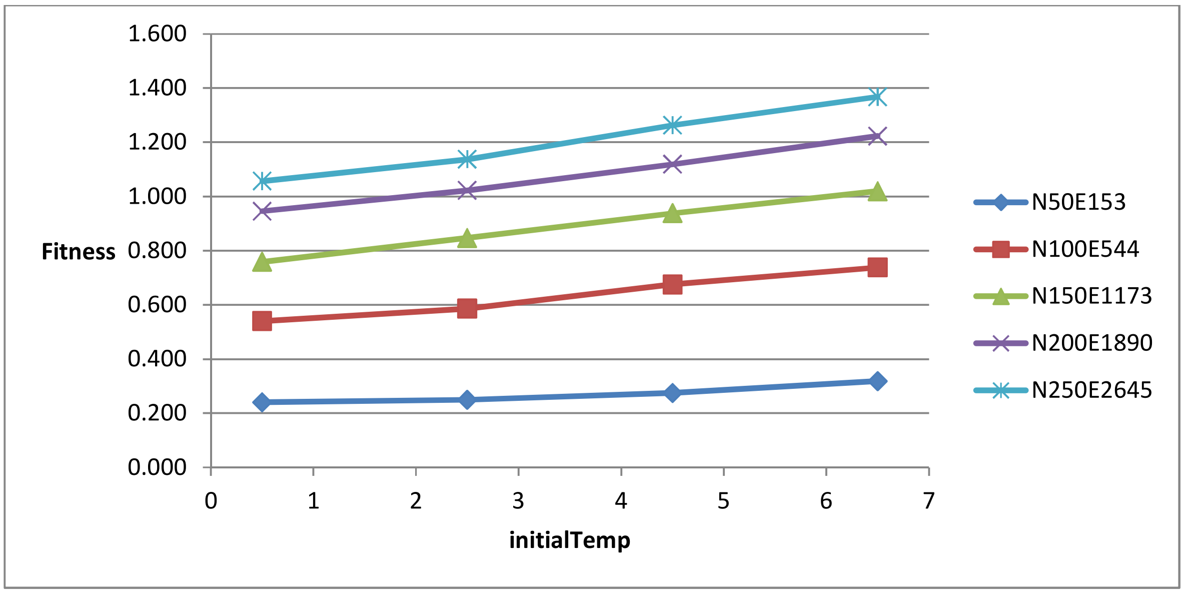


Figure 4.12 Simulated Annealing - Fitness values with the initialTemp parameter (phase I)

The cooling down parameter was tested with the values: 0.5, 0.6, 0.7, 0.8, 0.9. Figure 4.13 shows that there was no significant difference in the fitness values when *coolDown* was tested with the first four values. However, the value 0.9 gave a relatively poor graph layout compared to the other values. We chose the value 0.7 since it produced layouts of better fitness values when applied on large graphs.

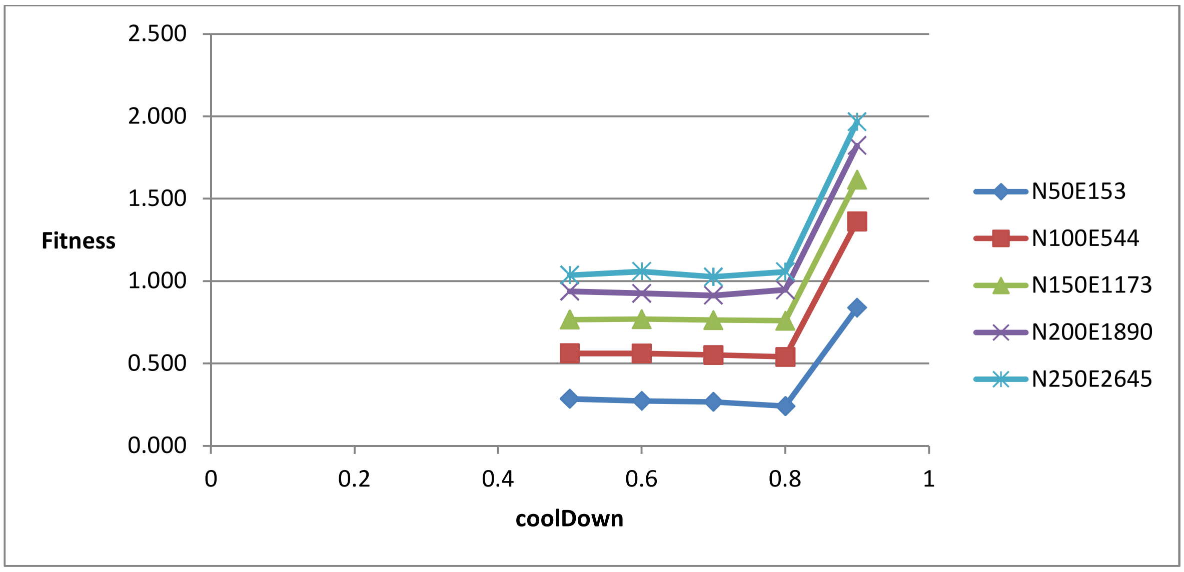


Figure 4.13 Simulated Annealing - Fitness values with the coolDown parameter (phase I)

ii. Phase II

Similar to phase I, we started the testing process with the *maxIterations* parameter by testing the values: 30, 35, 40, 45, 50. Figure 4.14 shows that the fitness values of the graph layouts became stable after 40 iterations for *maxIterations*. Therefore, we selected the first tested value after 40 which was the value 45, to become the value of this parameter.

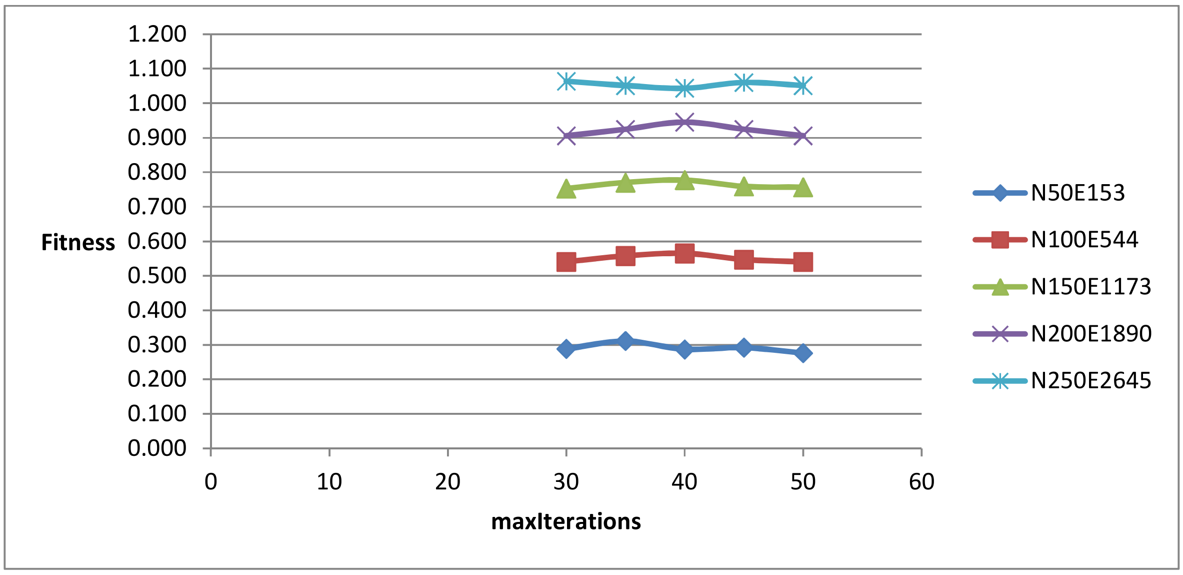


Figure 4.14 Simulated Annealing - Fitness values with the maxIterations parameter (phase II)

In the previous phase, we recognised that the higher the value of the *iterPerTemp* parameter, the better the quality of the produced layout. In this phase, we tested this parameter with the values: 10, 15, 20, 25, 30. The fitness values, as shown in Figure 4.15, were at their best when the value of *iterPerTemp* was either 25 or 30. The value 25 has been chosen since it produced very close fitness values to those generated when the value 30 was used. Furthermore, using the value 25 would make the algorithm generate a lower number of evaluated solutions compared to the number of solutions that would have been generated if the value 30 was used.

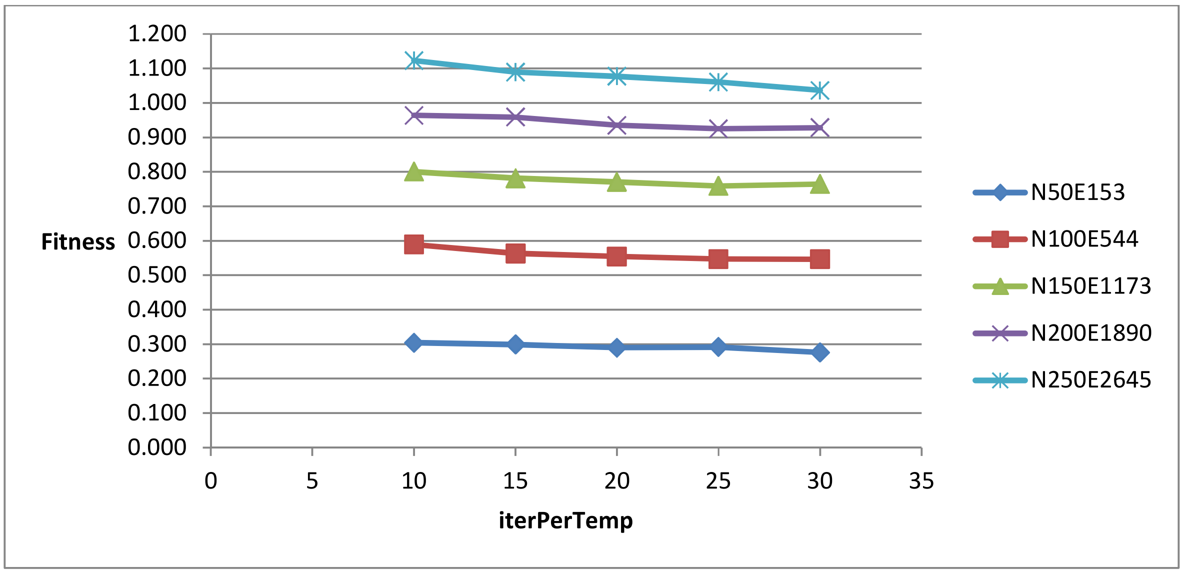


Figure 4.15 Simulated Annealing - Fitness values with the iterPerTemp parameter (phase II)

In phase I, we realised that increasing the value of the temperature parameter would result in producing layouts with poor quality. In this phase, we tested the *initialTemp* parameter with the values: 0.25, 0.5, 0.75, 1.0, 1.25. Unlike phase I, increasing the value of this parameter in phase II, has produced graph layouts with better quality compared to the values under test. Therefore, we can conclude that the value of this parameter should be below 2 (as shown in phase I testing) and above 1 (as shown in Figure 4.16). Although there is no major difference between the fitness values when the values 1.0 and 1.25 were used for the *initialTemp* parameter, we selected the value 1.25 as it produced slightly better solutions compared to those generated when the value 1.0 was used.

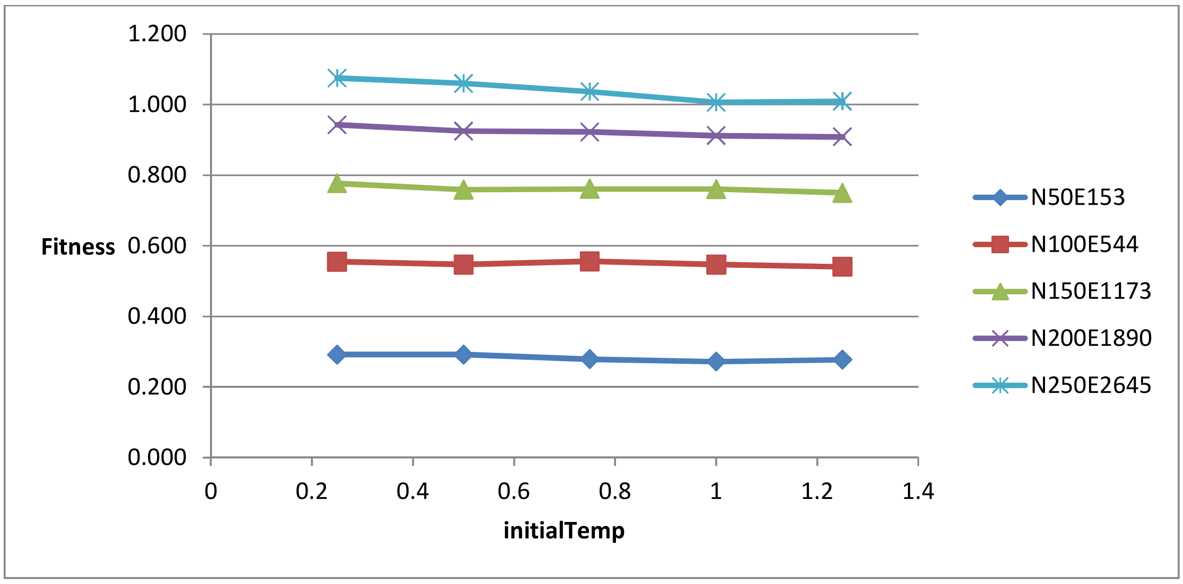


Figure 4.16 Simulated Annealing - Fitness values with the initialTemp parameter (phase II)

The *coolDown* parameter has been tested with the following values: 0.6, 0.65, 0.7, 0.75, 0.8. In phase I, using the value 0.9 for this parameter made the drawer produce layouts of low quality. That is why we selected a list of testing values that are below 0.9. According to Figure 4.17, the fitness values were relatively close but with an advantage of the fitness values (i.e. layouts) produced by the algorithm when the value of *coolDown* was 0.8. Therefore, we chose the value 0.8 for the *coolDown* parameter.

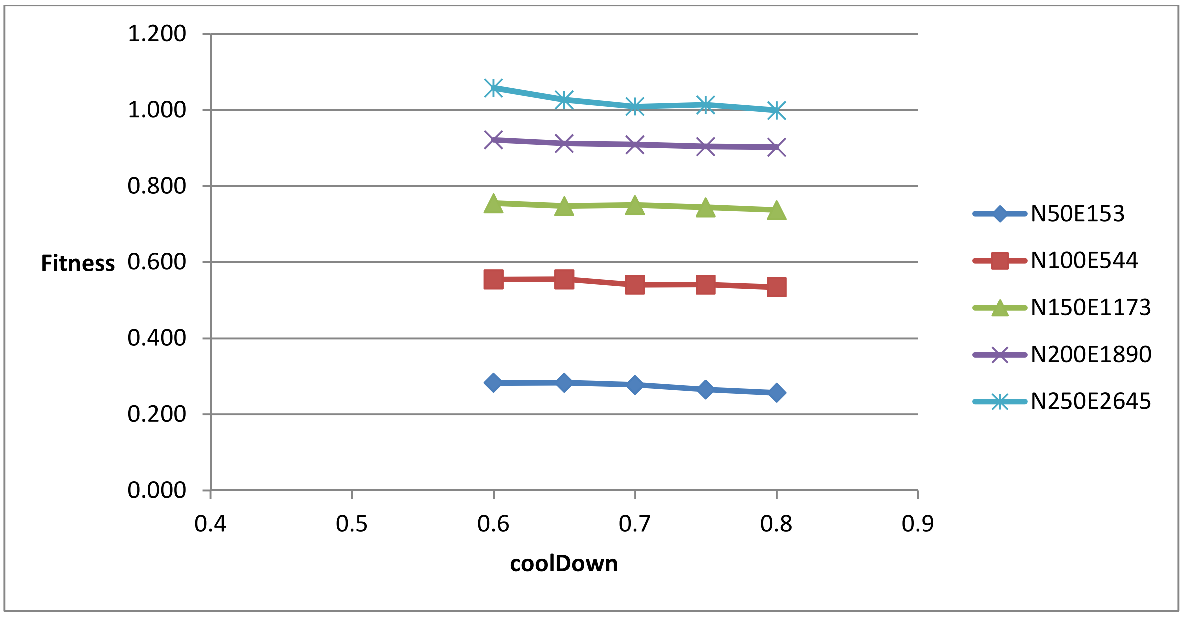


Figure 4.17 Simulated Annealing - Fitness values with the coolDown parameter (phase II)

iii. Phase III

Similar to phase II in Section 4.3.2, we took a view that a good-enough graph layout is a layout in which its fitness value is slightly greater than the best fitness value produced in the experiments of the previous phase. We used the values of the fitness function produced by the selected values of the parameters in phase II and we increased them by 12.5%. Then we ran the simulated annealing drawing algorithm until it reached equal fitness values to the target fitness values or no further improvement in the fitness value was made. Finally, we selected the most appropriate value for each parameter that gave a good-enough layout with a small number of evaluated solutions.

The main objective of this phase is speeding up the performance. The *maxIterations* parameter has a great effect on the number of evaluated solutions. Since 45 was the best value for this parameter in phase II, we selected values below 45 to test whether the algorithm can reduce the number of evaluated solutions and can still produce graphs with good-enough layouts. In this phase, we tested *maxIterations* with the following values: 25, 30, 35, 40, 45. According to Table 4.6, the values 40 and 45 were the only values that made the drawing algorithm produce graph layouts with fitness values that met the target fitness value. We selected the value 45 over the value 40, as Figure 4.18 shows that the number of evaluated solutions generated by the algorithm using the former value was lower than the number of evaluated solutions generated using the latter value as the graph size increases.

Table 4.6 Simulated Annealing - Fitness values with the maxIterations parameter (phase III)

	Fitness				
maxIterations	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
25	0.684	1.220	1.508	1.708	1.865
30	0.391	0.870	1.185	1.377	1.551
35	0.288	0.619	0.881	1.069	1.201
40	0.288	0.599	0.826	1.013	1.125
45	0.288	0.600	0.828	1.013	1.121

Target	0.289	0.601	0.829	1.015	1.124

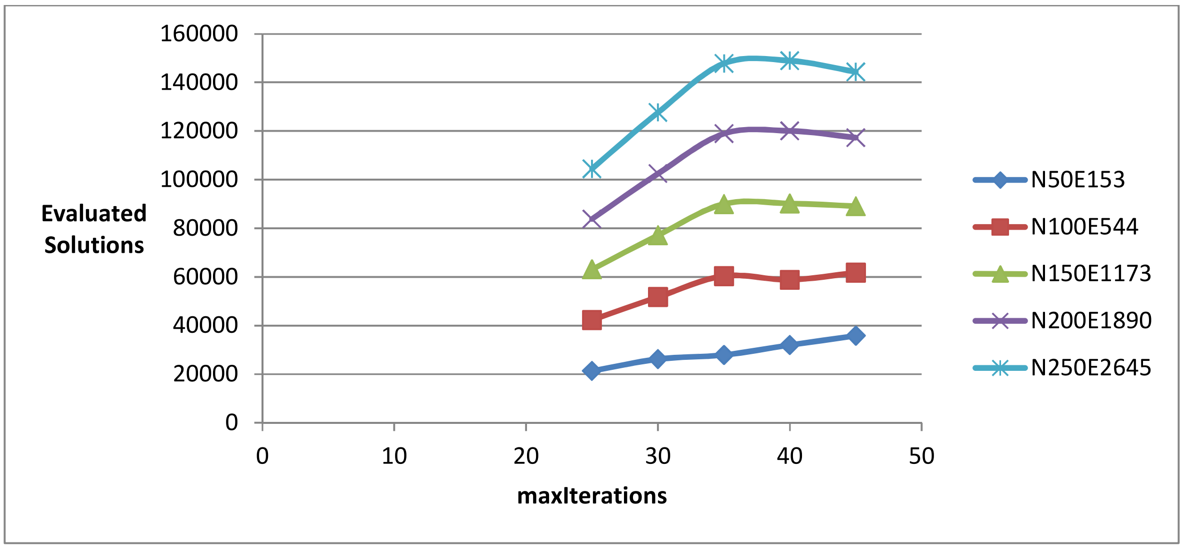


Figure 4.18 Simulated Annealing – Number of evaluated solutions with the maxIterations parameter (phase III)

In phase II, we ended up selecting the value 25 for the *iterPerTemp* parameter. Therefore, in phase III, we selected values which are less than 25 to test the possibility of using these values for producing graphs with good layouts and a few number of evaluated solutions. We tested this parameter with the values: 10, 15, 20, 25. Table 4.7 shows that all the values (except 10) had produced graph layouts with fitness values that met the target fitness values. We chose the value 15 for *iterPerTemp* since it generated a lower number of evaluated solutions compared to the values 20 and 25, as shown in Figure 4.19.

Table 4.7 Simulated Annealing - Fitness values with the iterPerTemp parameter (phase III)

	Fitness				
iterPerTemp	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
10	0.300	0.600	0.828	1.014	1.132
15	0.290	0.600	0.827	1.014	1.123
20	0.289	0.600	0.828	1.013	1.122
25	0.288	0.600	0.828	1.013	1.121

Target	0.289	0.601	0.829	1.015	1.124

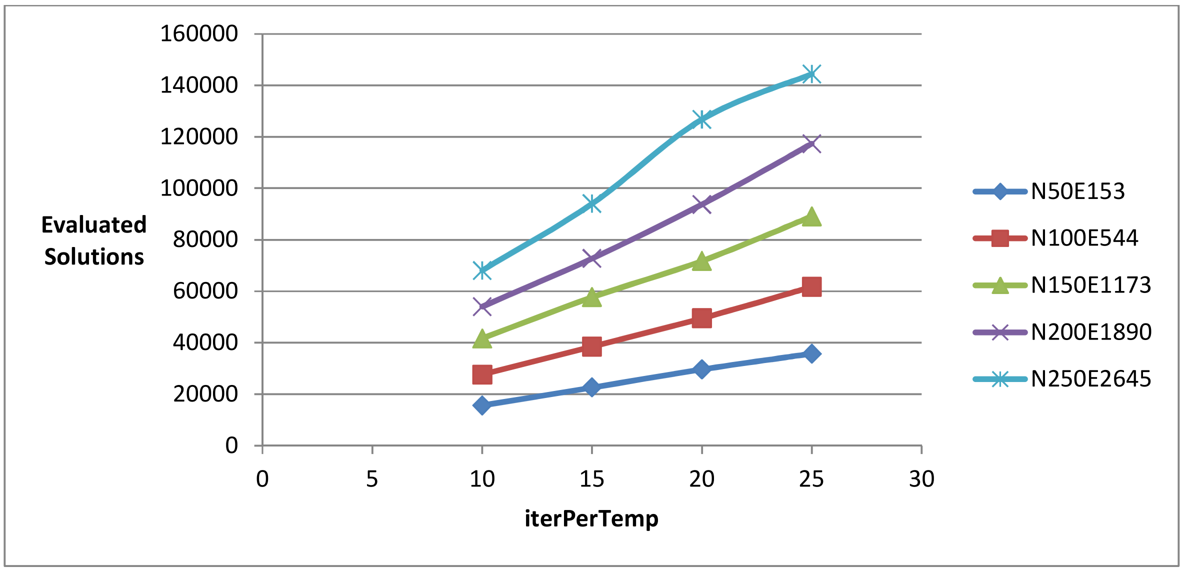


Figure 4.19 Simulated Annealing – Number of evaluated solutions with the iterPerTemp parameter (phase III)

The temperature parameter was tested with the values: 0.25, 0.75, 1.25, 1.75, 2.25. In Table 4.8, we can see that using any of these values would give graph layouts with fitness values that meet the target fitness values. On the other hand, Figure 4.20 shows that there was no clear behaviour for the number of evaluated solutions before the value 0.75. But starting from this value onwards, the figure shows that the number of evaluated solutions increased as a function of the graph size. Thus, we selected the value 0.75 for the *initialTemp* parameter.

Table 4.8 Simulated Annealing - Fitness values with the initialTemp parameter (phase III)

	Fitness				
initialTemp	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
0.25	0.314	0.601	0.828	1.015	1.113
0.75	0.299	0.601	0.828	1.014	1.117
1.25	0.290	0.600	0.827	1.014	1.123
1.75	0.290	0.600	0.827	1.013	1.122
2.25	0.288	0.600	0.827	1.012	1.122
Target	0.289	0.601	0.829	1.015	1.124

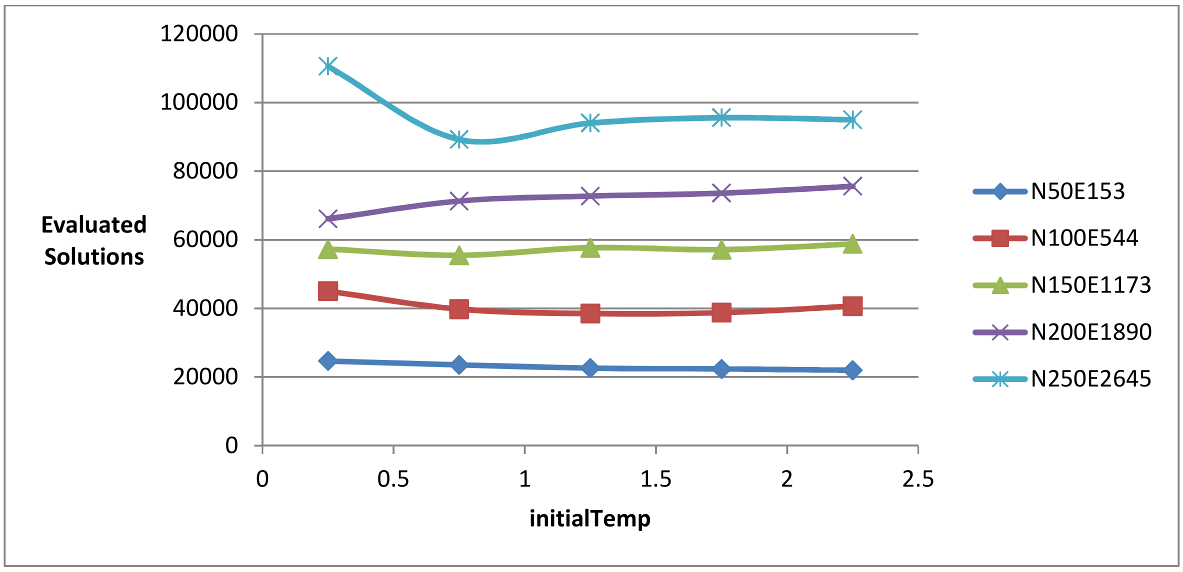


Figure 4.20 Simulated Annealing – Number of evaluated solutions with the initialTemp parameter (phase III)

The behaviour of the *coolDown* parameter was not very clear in this phase, but it was still possible to take a decision for the most appropriate value for this parameter. Table 4.9 shows that all the values which we tested for *cooldown*: 0.65, 0.7, 0.75, 0.8, 0.85, would give graph layouts with fitness values that meet the target fitness values (excluding the 1st set of graphs). Figure 4.21 does not illustrate a clear behaviour of the effect of this parameter on the number of evaluated solutions. However, using the value 0.8 for *coolDown* had generated a lower number of evaluated solutions (except for the 4th set of graphs) compared to all the other values under test. Therefore, we chose the value 0.8 for this parameter.

Table 4.9 Simulated Annealing - Fitness values with the coolDown parameter (phase III)

	Fitness				
coolDown	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
0.65	0.306	0.601	0.828	1.015	1.106
0.7	0.302	0.600	0.828	1.012	1.119
0.75	0.301	0.600	0.828	1.014	1.125
0.8	0.299	0.601	0.828	1.014	1.117
0.85	0.293	0.622	0.889	1.119	1.278

Target	0.289	0.601	0.829	1.015	1.124

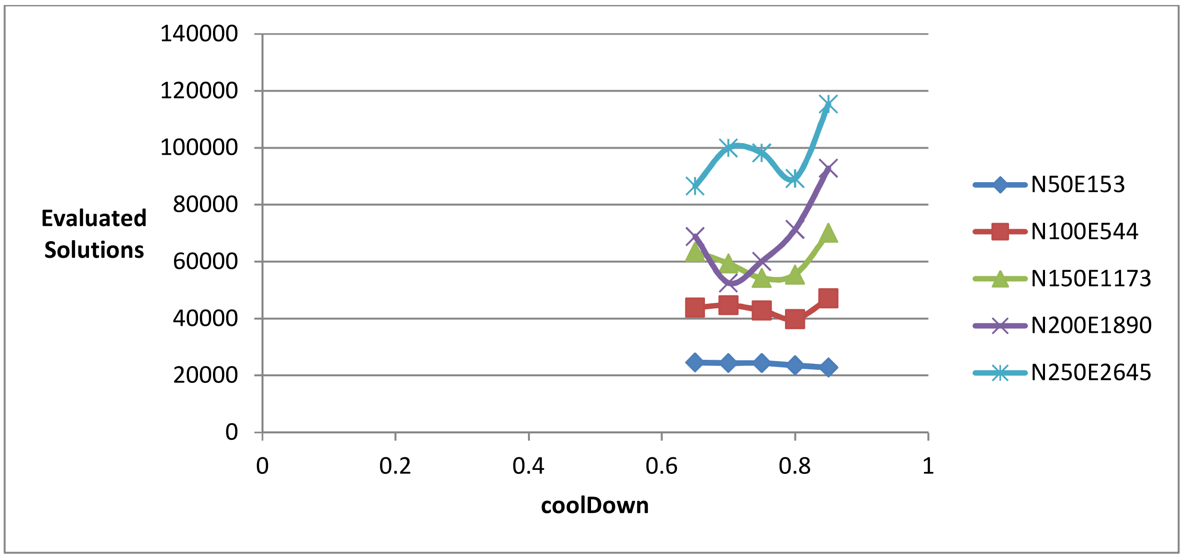


Figure 4.21 Simulated Annealing – Number of evaluated solutions with the coolDown parameter (phase III)

4.5 Tabu Search

Tabu search is a neighbourhood search-based approach that uses a memory structure while it carefully explores the neighbourhood of each solution as the search progresses to avoid getting trapped in local optima. It proceeds on the assumption that there is no value in choosing an inferior solution unless it is necessary, as in the case of escaping from a local optimum (Lim & Chee 1991). It improves the efficiency of the searching process by storing a tabu list of local solutions. This is used to restrict the search by forbidding moves to some poor neighbour solutions that have already been visited (Hertz et al. 1995). An additional feature of tabu search is applying intensification and diversification. It might be useful to intensify the exploration in some region because it may contain a high incidence of acceptable solutions. This can be obtained by introducing a new term in the objective function that assigns a high priority to solutions in the relevant region. Diversification is responsible for moving the exploration process over different regions of the search space (Marti 1998).

Our tabu search algorithm goes through a predefined number of iterations to minimise the value of the fitness function. It uses a tabu list to store tabu moves in order to prevent the algorithm from choosing previously reached moves for particular nodes for a predefined period of time. Algorithm 4.3 represents the steps of our tabu search method.

4.5.1 Algorithm

In outline, as described in the algorithm, the tabu search method operates in the following manner: first, we compute the fitness value of the initial layout. Then the following steps are performed for a set number of iterations (`maxIterations`): for each node, we search in the neighbourhood for candidate solutions, as described in section 4.2.1. The ratio of the fitness value of the candidate solution to the fitness value of the current solution is computed at each point in the neighbourhood. Solutions with ratios above or equal to a predefined threshold value (`initialCutOff`) are considered to be tabu moves and are stored in a tabu list. We then move the node to a neighbouring point that is not in the tabu list and its fitness function value is minimum compared to all neighbours. Then the current solution is added to the tabu list. Note that the new solution might not be better than the current solution hence the tabu search does not run out of solutions. In case all eight candidate solutions surrounding the current solution are in the tabu list, the intensification and the diversification processes will be the way out for solving this problem. A search intensification process is implemented: after a chosen number of iterations (`intensifyIterations`), the square size centred on the node is reduced and the cut-off value is decreased by a set value (`intensifyCutOff`) by calling function `SmallerSquareSize()` and function `SmallerTabuCutOff()` respectively, as shown in Algorithm 4.3. Finally, in order to diversify the searching space, the tabu list is updated by removing old solutions from the list after a number of iterations (`duration`).

Given:

Connected Graph `G(V,E)`: V is a set of nodes and E ⊆ (V×V) is a set of edges.

`initialSquareSize`: predefined size of a square where candidate solutions are located on its border.

`squareReduction`: predefined value which represents the rate of reduction for the size of the square.

`maxIterations`: predefined maximum number of iterations of the drawer.

`initialCutOff`: predefined minimum value that determines whether a move is tabu or not.

`intensifyCutOff`: predefined value which represents the rate of reduction for *`cutOff`*.

`intensifyIterations`: predefined number of iterations in which the searching process starts to intensify.

`duration`: predefined number of iterations in which a move should remain in the tabu list.

Algorithm:

```
1: allOffsets = {(1,1), (1,0), (1,-1), (0,-1), (-1,-1), (-1, 0), (-1, 1), (0, 1)}
2: tabuSet = {}
3: squareSize = initialSquareSize , CutOff = initialCutOff
4: layout = RandomizeLayout(G) /* layout maps each node in G to an (x,y) position */
5: iteration = 0
6: while iteration < maxIterations do
7:  for v in V do
8:   currentPos = layout[v] /* position currently associated with node v */
9:  currentFitness = Fitness(layout)
10:  candidates = {}
11:  for scaledOffset in {(squareSize*x, squareSize*y) | (x,y) in allOffsets}
12:   candidatePos = currentPos + scaledOffset /* vector addition */
13:   if (v, candidatePos, i) ∉ tabuSet for some i then
14:    layout[v] = candidatePos
15:    candidateFitness = Fitness(layout)
16:    if candidateFitness / currentFitness > CutOff then
17:     tabuSet = tabuSet ∪ {(v, candidatePos, iteration)}
18:    else
19:     candidates = candidates ∪ {(candidatePos, candidateFitness)}
20:    end if
21:   end if
22:  end for
23:  if candidates ≠ {} then
24:   newPos = p, where (p,f) is the pair in candidates with minimal f
25:   layout[v] = newPos
26:   tabuSet = tabuSet ∪ {(v, currentPos, iteration)}
27:  end if
28: end for
29: if (iteration mod intensifyIterations) == 0 then
30:  squareSize = SmallerSquareSize(squareSize, squareReduction)
31:  CutOff = SmallerTabuCutOff(CutOff, intensifyCutOff)
32: end if
33: tabuSet = {(v,p,i) | (v,p,i) in tabuSet and (iteration - i) < duration}
34: iteration = iteration + 1
35:end while
```

Algorithm 4.3 Our tabu search graph drawing algorithm

Note that the `SmallerSquareSize()` function reduces the square size used in searching for candidate solutions by applying the same formula we used in hill climbing. Whereas the `SmallerTabuCutOff()` function decreases the value of cut-off during the intensification process to maintain high quality candidate solutions and truncate the other solutions by adding them to the tabu list. The function uses the following formula for cut-off value reduction, such that the initial value of `oldCutOff` is equal to *`initialCutOff`*:

newCutOff = oldCutOff – (*intensifyCutOff* * *intensifyIterations)*

4.5.2 Parameter Tuning

Tabu search has five parameters that affect the quality of the layouts produced by the algorithm along with its performance: the total number of iterations needed for execution (*maxIterations*), the cut-off value which determines whether to consider a solution for further testing or to add it to the tabu list (*initialCutOff*), the value used in decreasing the cut-off value for intensifying the search process (*intensifyCutOff*), the number of iterations required to decrease the value of the cut-off (i.e. intensify the search) (*intensifyIterations*), and the duration in which a solution remains in the tabu list (*duration*).

The graph sets used in testing the values of these parameters were exactly the same sets used in testing the values of the parameters of hill climbing, as described earlier in this chapter in Table 4.1.

Tabu search parameters are dependent. Therefore, we followed the same incremental testing process that we performed with simulated annealing but we divided the process into four phases. In phase I, we considered the values that gave good graph layouts (small fitness values) regardless of the number of evaluated solutions performed by the drawing algorithm. In the second phase of parameter tuning, we repeated the same steps followed in phase I, but instead of starting with arbitrary values, we started with the values that were selected and fixed from phase I. Moreover, we narrowed the differences between the tested values for each parameter. In phase III, we tested the effect of the values of tabu search parameters on the performance of the drawing algorithm (i.e. number of evaluated solutions). In all the previous phases, we used an initial square size of 256. However, after performing a complete testing on the parameters of hill climbing, an initial square size of 512 has produced graph layouts with better quality and a lower number of evaluated solutions were performed by the drawing algorithm. Therefore, in phase IV, we repeated the same tuning process that we performed on the tabu search drawing algorithm in phase III using the best initial square size as described in the hill climbing parameter tuning process.

i. Phase I

We tested the values of *maxIterations* and fixed the values of the other parameters to some arbitrary values such that *initialCutOff =2*, *intensifyCutOff = 0.005*, *intensifyIterations = 5*, and *duration = 5*. The values used in testing *maxIterations* were: 30, 40, 50, 60, 70. According to Figure 4.22, the values 50 and 60 produced the best fitness values compared to others with an advantage to the value 50 as the graph size becomes larger. Thus, we selected the value 50 for *maxIterations.*

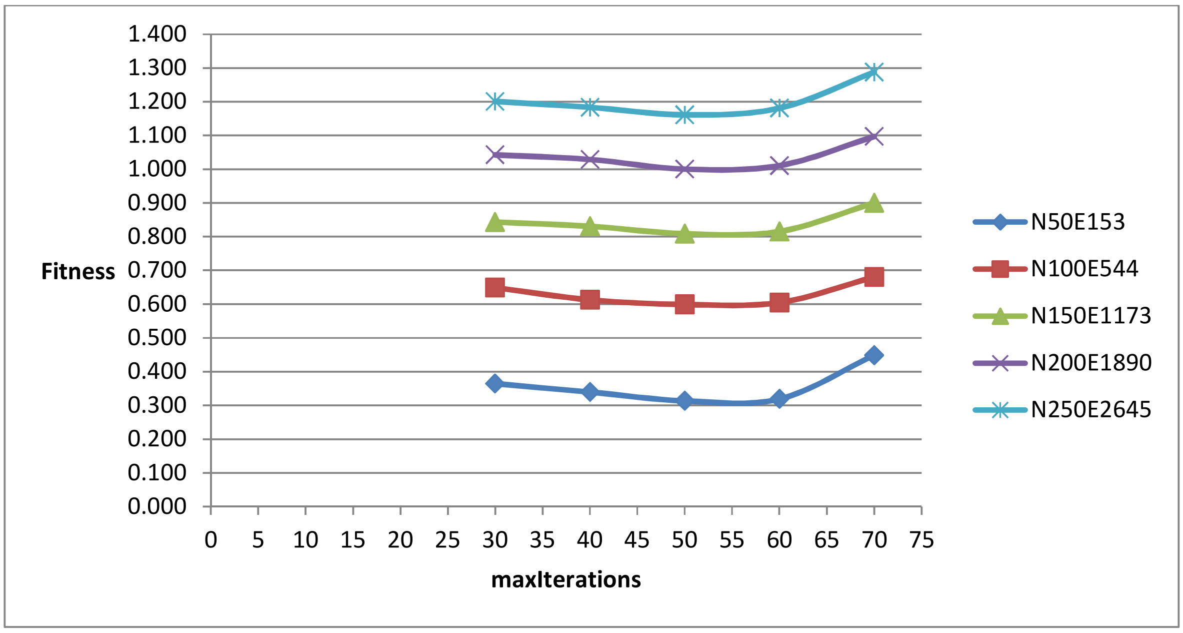


Figure 4.22 Tabu Search - Fitness values with the maxIterations parameter (phase I)

Secondly, after fixing the value of *maxIterations*, we moved on to test the value of the *initialCutOff* parameter and we kept the rest of the parameters with their arbitrary values. *initialCutOff* has been tested with the following values: 0, 2, 4, 6, 8, 10. We chose the value 0 to see the effect of increasing the number of tabu solutions on the quality of the produced layouts. Figure 4.23 shows the effect of the *initialCutOff* values on the fitness values of the graph sets. According to the figure, the fitness function values look similar when the *initialCutOff* value is between 2 and 10. However, we selected the value 4 since it produced slightly better fitness values compared to 2 and almost the same as the rest of the values except 0.

Now that we fixed the values of two parameters, we moved to the third parameter *intensifyCutOff* and tested it with the following values: 0.005, 0.055, 0.105, 0.155, 0.205, while keeping the rest of the parameters as they were. Figure 4.24 shows that the fitness values are very close when the value of this parameter is between 0.005 and 0.055 with an advantage to 0.005 for graphs with smaller sizes. Therefore, we selected the value 0.005 for *intensifyCutOff*.

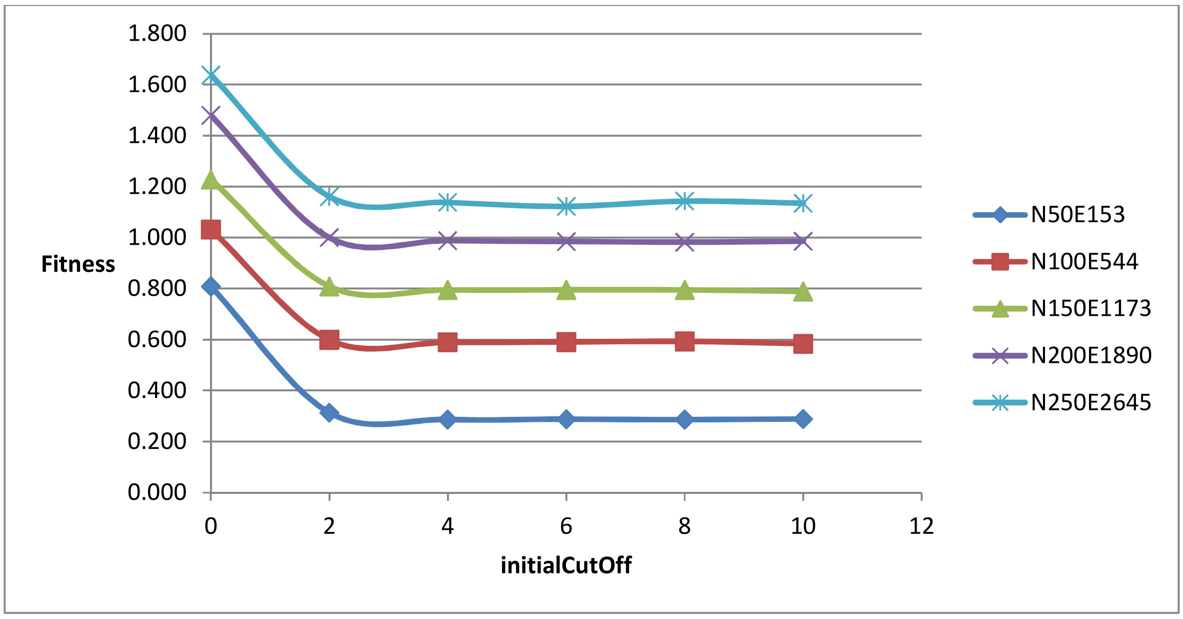


Figure 4.23 Tabu Search - Fitness values with the initialCutOff parameter (phase I)

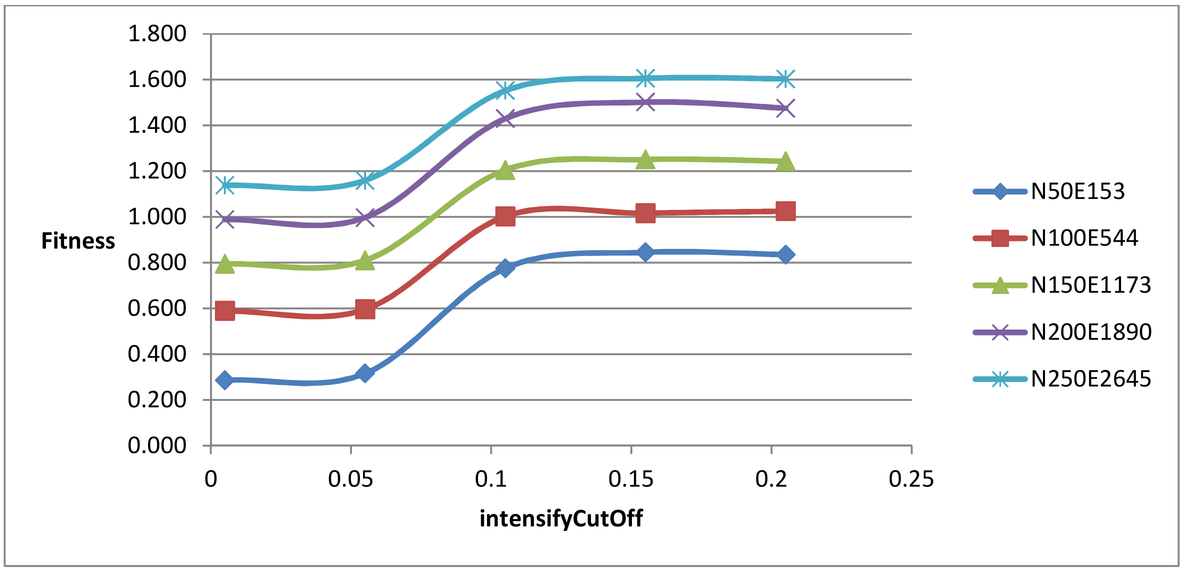


Figure 4.24 Tabu Search - Fitness values with the intensifyCutOff parameter (phase I)

intensifyIterations was the next parameter to be tested after fixing the values of three parameters. It has been tested with the following selected values: 1, 3, 5, 7, 9. This parameter shows the effect of the number of iterations required to reduce the value of the cut-off. Figure 4.25 shows that there is no significant difference between the values selected, but the curve starts to increase slightly after the value 5. That means that increasing the value of *intensifyIterations* would produce low-quality graph layouts. This is normal, since the intensification process should take place after a reasonable but not a large number of iterations taking into account that there is a limited number of iterations for the algorithm to execute (*maxIterations)*. For this reason, we selected the value 5 for *intensifyIterations*. Another reason for choosing this value, not a smaller one, was that the number of accesses to the tabu list is higher with the value 5 compared to the values 1 and 3, as shown in Figure 4.26, and the higher the number of accesses to the tabu list, the lower the number of evaluated solutions as more solutions would be excluded from the searching process.

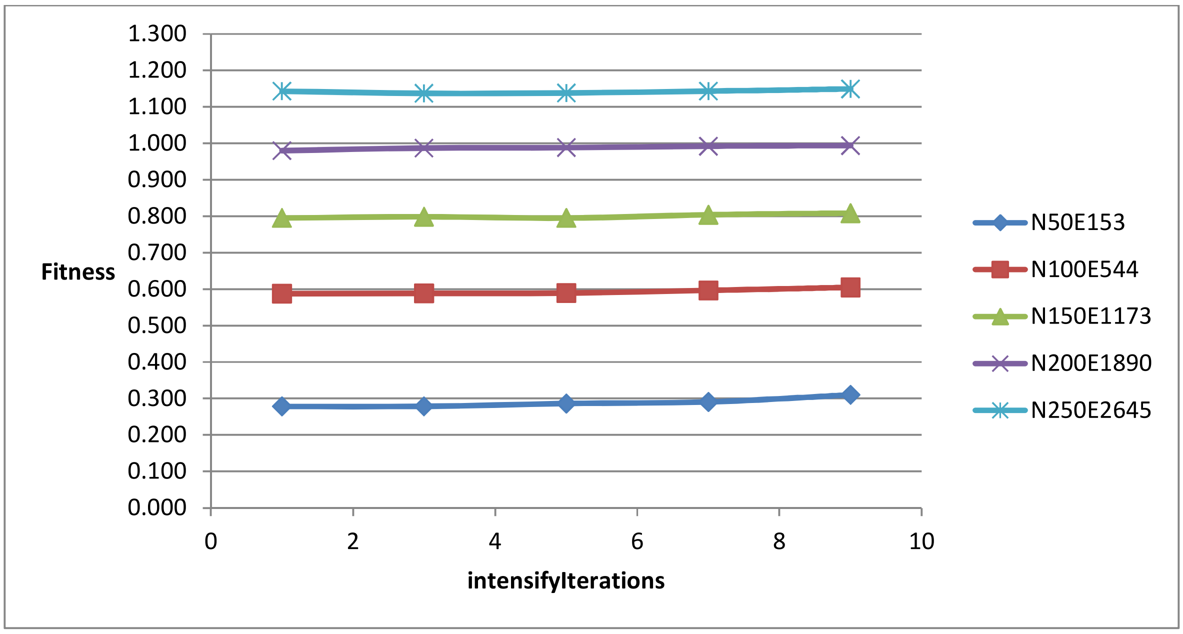


Figure 4.25 Tabu Search - Fitness values with the intensifyIterations parameter (phase I)

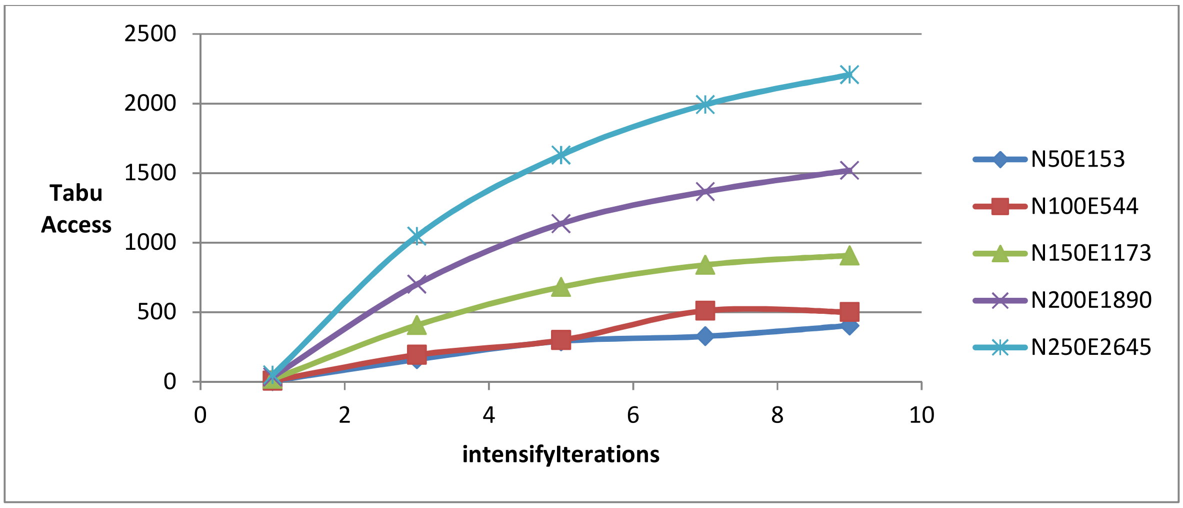


Figure 4.26 Tabu Search - tabu list accesses with the intensifyIterations parameter (phase I)

The last parameter that has been tested in this phase was *duration*. We tested this parameter with the following values: 0, 5, 15, 25, 35, while all the other parameters were fixed. Figure 4.27 shows that there is no significant effect of this parameter on the fitness value of the produced graph layouts. However, Figure 4.28 shows that number of accesses to the tabu list is small when the duration is below 5 and consequently, the number of evaluated solutions would increase. On the other hand, the performance of the drawing algorithm looks stable after the value 5. Therefore, we selected the value 5 for this parameter.

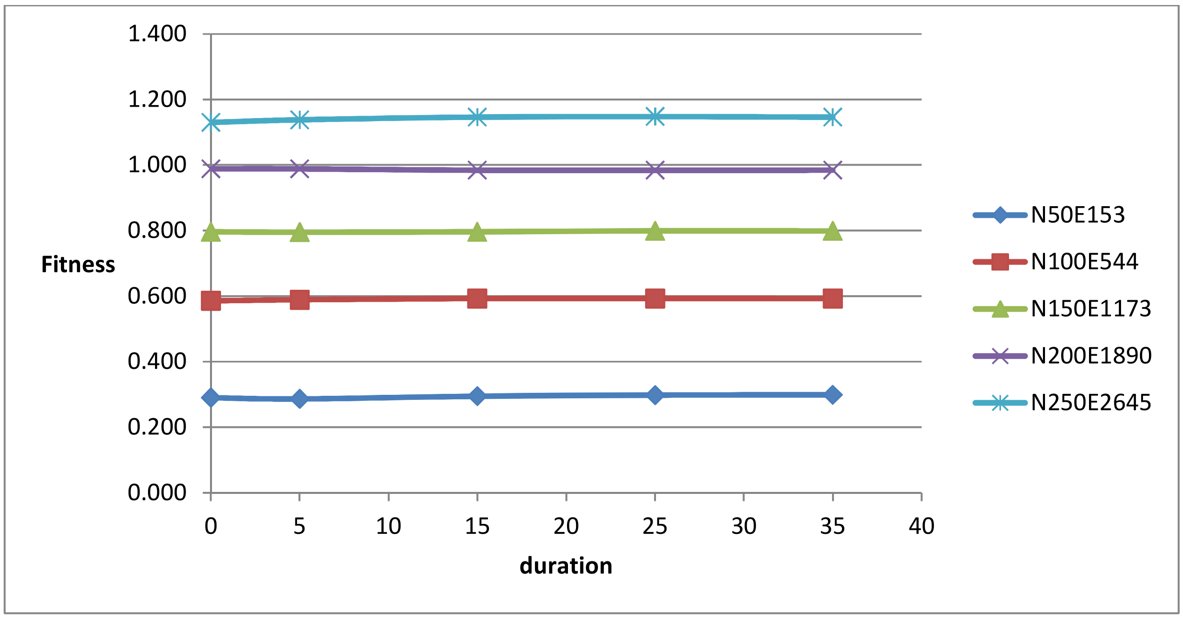


Figure 4.27 Tabu Search - Fitness values with the duration parameter (phase I)

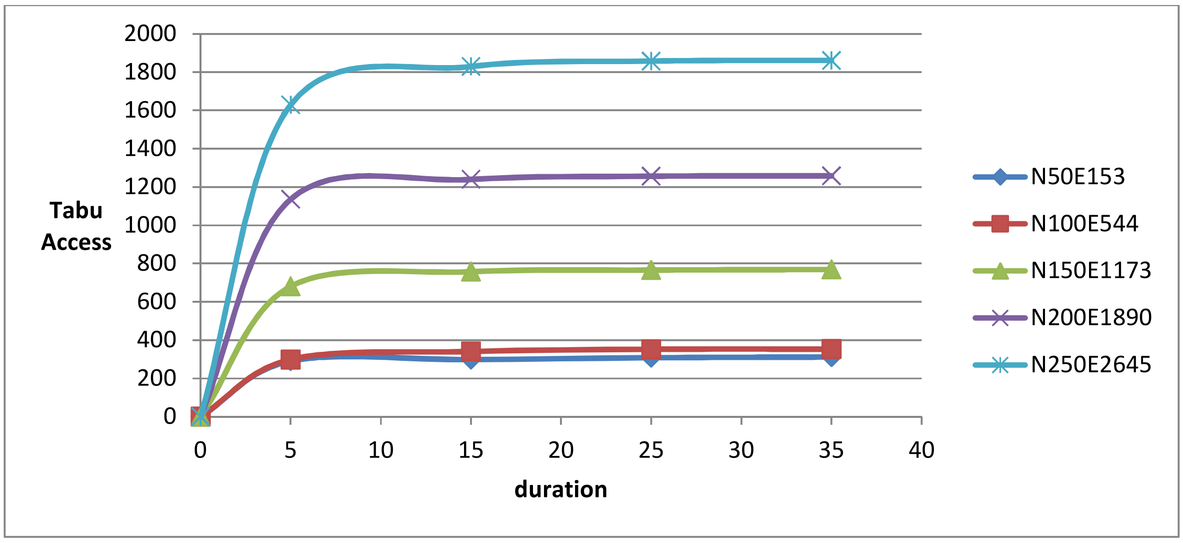


Figure 4.28 Tabu Search - tabu list accesses with the duration parameter (phase I)

ii. Phase II

We narrowed the differences between the tested values for each parameter. For example, in phase I, the difference between the values we tested for *maxIterations* was 10. In this phase, we reduced the difference to 5. The best value we got for *maxIterations* in phase I was 50. Now, we tested this parameter with the following values: 40, 45, 50, 55, 60. As in phase I, we chose the value which gives the best fitness value regardless of the number of evaluated solutions produced by the drawer. With reference to Figure 4.29, the best value for *maxIterations* is 55.

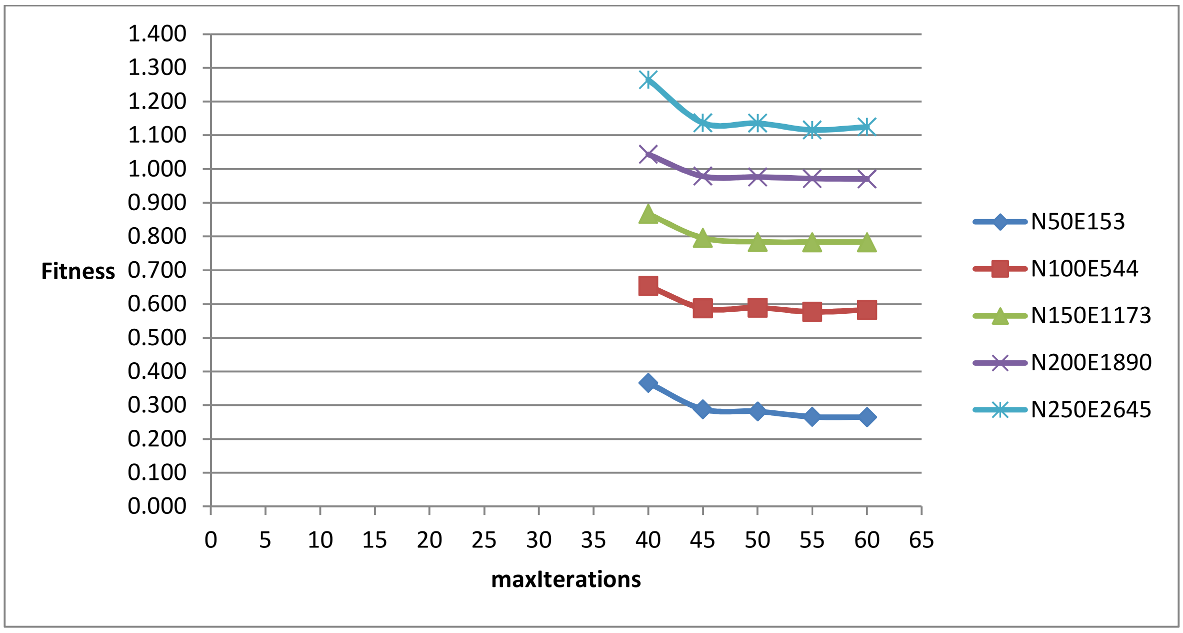


Figure 4.29 Tabu Search - Fitness values with the maxIterations parameter (phase II)

As for the *initialCutOff* parameter, we recognised in the first phase that there is no significant difference in the fitness value when we tested the *initialCutOff* with several values except for the value 0. In this phase, we got similar results. The fitness value reduces as we increased the value of the *initialCutOff*. Nevertheless, the reduction rate was barely recognised. Figure 4.30 shows the effect of the different values we tested for this parameter: 2, 3, 4, 5, 6, 7, 8, on the fitness value. According to the figure, all the values of *initialCutOff* gave very close values for the fitness function. However, the *initialCutOff* value 7 gave a slightly better fitness value compared to the others.

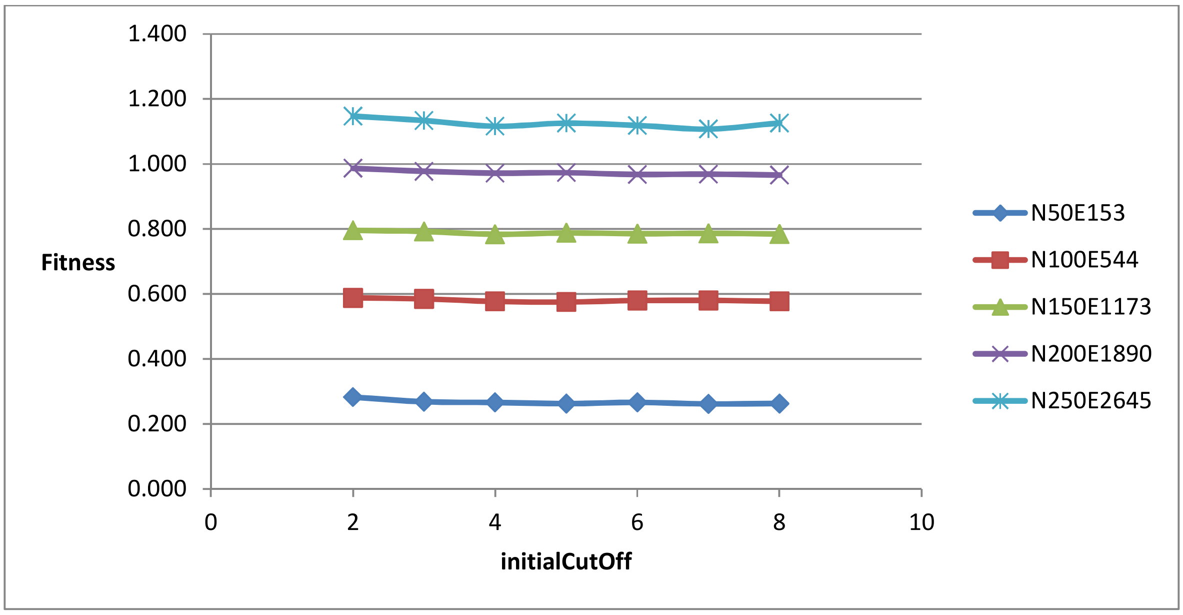


Figure 4.30 Tabu Search - Fitness values with the initialCutOff parameter (phase II)

At the end of phase I, we fixed the value of *intensifyCutOff* to 0.005. We recognised that the fitness value was better when the value of this parameter was below 0.1. Therefore, in this phase, we tested this parameter with values less than 0.1 such as: 0.005, 0.025, 0.045, 0.065, 0.085. However, Figure 4.31 shows that any of these values could be selected as a value for this parameter since there was no major difference between the fitness values. Thus, we kept the same value that we selected in phase I which equals to 0.005.

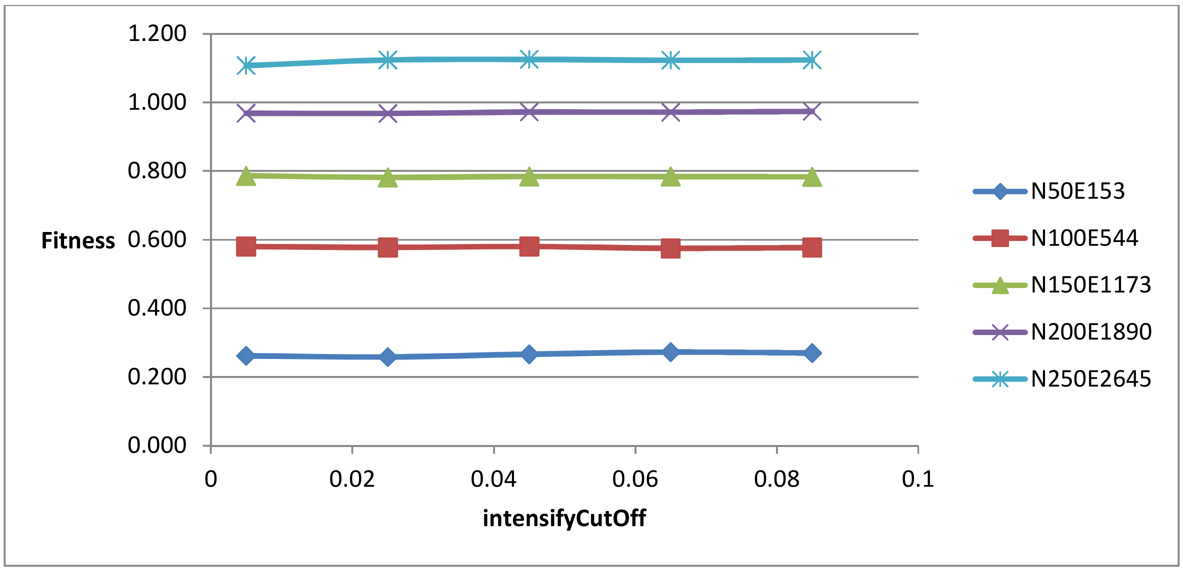


Figure 4.31 Tabu Search - Fitness values with the intensifyCutOff parameter (phase II)

The results of phase I showed that increasing the value of *intensifyIterations* would also increase the value of the fitness function (i.e. reduce the quality of the graph layout). In this phase, we selected the following values for testing: 3, 5, 7, 9, 11. Figure 4.32 shows that the best values for the fitness function were produced when the value of *intensifyIterations* was 5 (the same value we selected in phase I). Furthermore, the figure confirmed the fact that increasing the value of this parameter would reduce the quality of the graph layout.

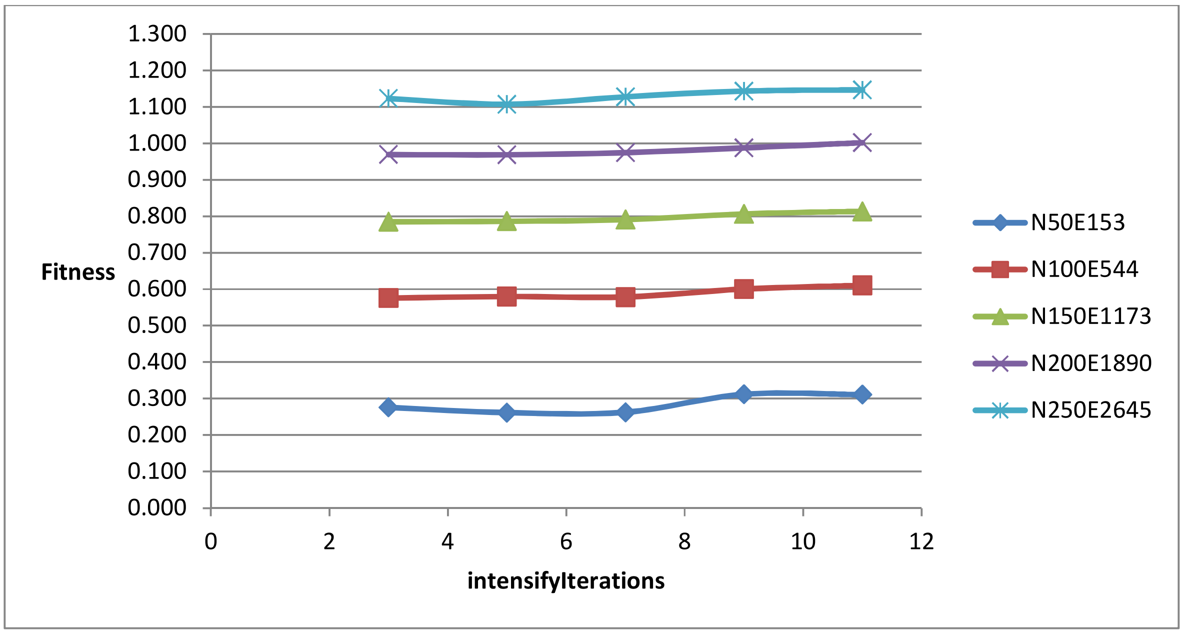


Figure 4.32 Tabu Search - Fitness values with the intensifyIterations parameter (phase II)

As shown in the previous phase, increasing the value of *duration* starting from the value 5 would not make any significant changes in the values of the fitness function. This is what we got when we tested this parameter again in phase II with the values: 5, 15, 25, 35, 45, as shown in Figure 4.33. Therefore, we have not made any changes to the value of *duration* and kept the fixed value from the previous phase which was 5. However, the value of this parameter slightly affects the number of evaluated solutions generated by the drawer as we will see in the next phase.

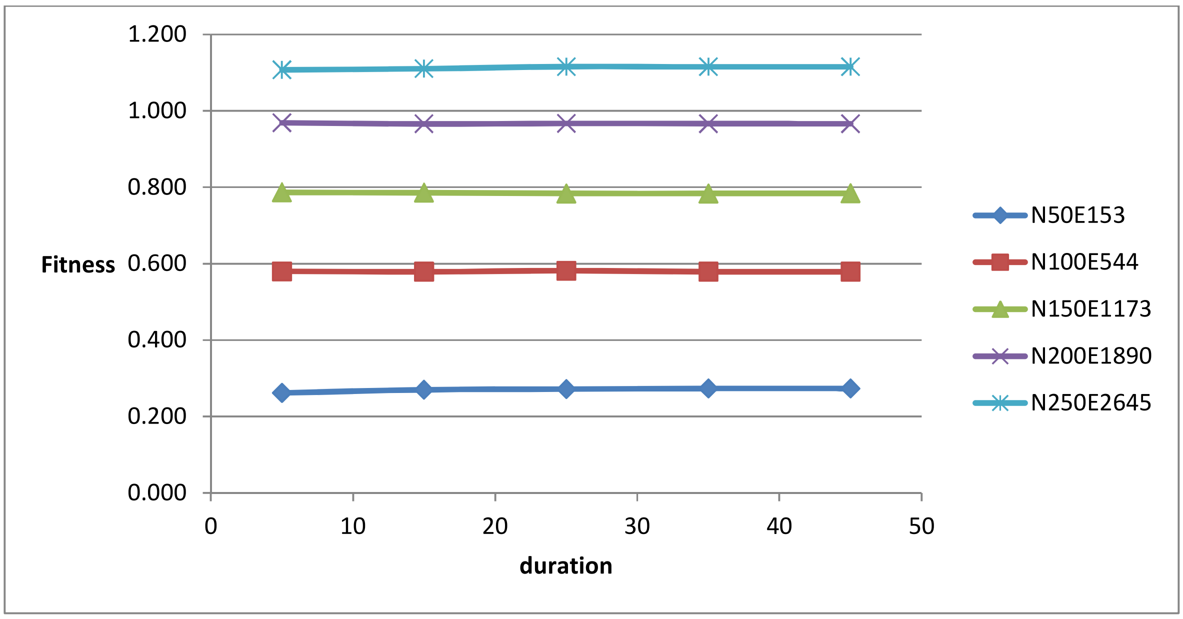


Figure 4.33 Tabu Search - Fitness values with the duration parameter (phase II)

iii. Phase III

We performed a similar procedure to the one we used in parameter tuning in phase II of hill climbing and in phase III of simulated annealing. We took a view that the fitness value of a good-enough graph layout can be slightly larger than the best fitness value produced in the experiments of the previous phase. We used the values of the fitness function produced by the selected values of the parameters in phase II and we increased them by 12.5%. Then we ran the tabu search drawing algorithm until it reached equal fitness values to the target fitness values or no further improvement on the fitness value was made. Finally, we picked the most appropriate values of the parameters that gave a good-enough layout with a small number of evaluated solutions.

As we are looking to minimise the number of evaluated solutions, we tuned the value of *maxIterations* by testing the following values: 35, 40, 45, 50, 55. Since the value 55 was the best value we got in phase II, we tested this parameter with values lower than 55 to see whether we can get a good layout with a lower number of iterations. Table 4.10 shows the fitness values produced by the tabu search drawing algorithm compared to the target fitness values. On the other hand, Figure 4.34 shows the number of evaluated solutions produced by our tabu search drawing algorithm. According to the results in the table and the figure, we selected the value 45 for *maxIterations* since it produced fitness values that are similar or less than the target fitness values with a low number of iterations compared to the other values.

Table 4.10 Tabu Search - Fitness values with the maxIterations parameter (phase III)

	Fitness				
maxIterations	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
35	0.344	0.652	0.876	1.073	1.234
40	0.370	0.654	0.872	1.071	1.245
45	0.293	0.641	0.868	1.078	1.239
50	0.302	0.640	0.869	1.078	1.235
55	0.287	0.628	0.875	1.078	1.238
Target	0.294	0.652	0.885	1.090	1.246

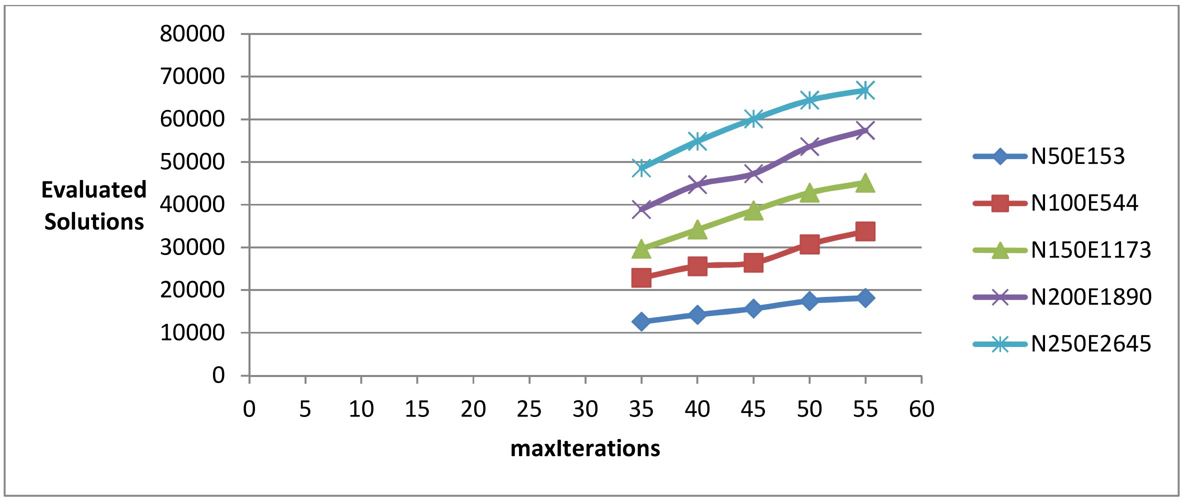


Figure 4.34 Tabu Search – Number of evaluated solutions with the maxIterations parameter (phase III)

As shown in the previous phases, increasing the value of the *initialCutOff* slightly reduces the value of the fitness function. On the other hand, in this phase, the experiment showed that increasing the value of this parameter would slightly increase the number of evaluated solutions. The best value we got for this parameter in phase II was 7. Therefore, we tested it with lower values: 1, 2, 3, 4, 5 in order to verify whether we can obtain a good layout with a small number of evaluated solutions. Table 4.11 and Figure 4.35 indicate that the value 2 could be the best value for *initialCutOff* since the number of evaluated solutions became stable starting from that value.

Table 4.11 Tabu Search - Fitness values with the initialCutOff parameter (phase III)

	Fitness				
initialCutOff	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
1	0.821	0.996	1.215	1.398	1.597
2	0.314	0.645	0.866	1.069	1.235
3	0.298	0.643	0.867	1.074	1.244
4	0.302	0.640	0.868	1.072	1.238
5	0.295	0.643	0.867	1.077	1.240
Target	0.294	0.652	0.885	1.090	1.246

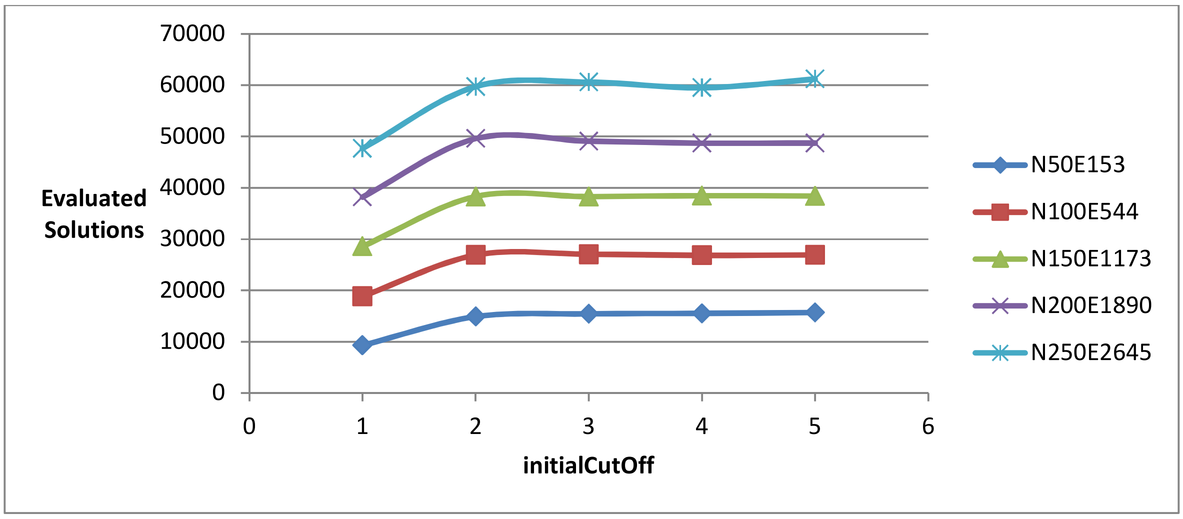


Figure 4.35 Tabu Search – Number of evaluated solutions with the initialCutOff parameter (phase III)

For the *intensifyCutOff* parameter, we tuned the value by testing it with values close to the value 0.005 (as selected in phase II). The values which we tested were: 0.0025, 0.005, 0.0075, 0.01, 0.0125. Table 4.12 shows that our drawing algorithm produced fitness values similar or lower than the target values with all the tested values (except for the first set of graphs). Furthermore, the number of evaluated solutions is almost similar among all the tested values with a minor advantage for the value 0.0025 in the first four sets of the graphs, as shown in Figure 4.36. Therefore, we picked the value 0.0025 for the *intensifyCutOff* parameter.

Table 4.12 Tabu Search - Fitness values with the intensifyCutOff parameter (phase III)

	Fitness				
intensifyCutOff	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
0.0025	0.313	0.643	0.870	1.068	1.230
0.005	0.314	0.645	0.866	1.069	1.235
0.0075	0.329	0.648	0.869	1.072	1.236
0.01	0.330	0.645	0.869	1.072	1.230
0.0125	0.331	0.647	0.870	1.077	1.235
Target	0.294	0.652	0.885	1.090	1.246

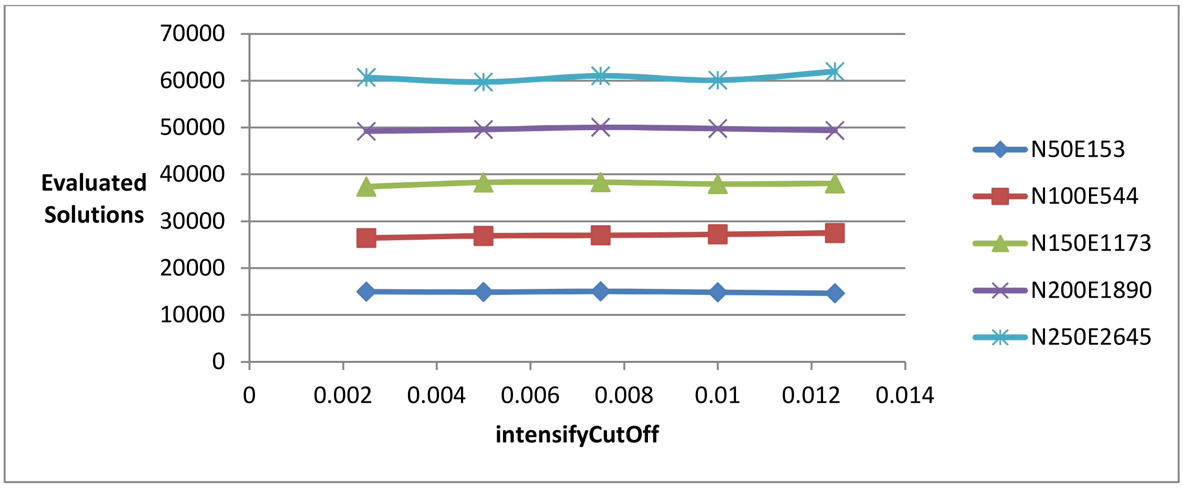


Figure 4.36 Tabu Search – Number of evaluated solutions with the intensifyCutOff parameter (phase III)

The fourth parameter, *intensifyIterations*, was tested using the following values: 3, 5, 7, 9, 11. According to Table 4.13, the values 5 and 7 gave smaller fitness values (better quality) compared to the other values of the parameter as the graph size increased. Whereas, the number of evaluated solutions produced by the algorithm when the value of this parameter is 5, is smaller than or equal to the number of evaluated solutions given by the algorithm using the rest of the values (except for the fourth set of graphs), as shown in Figure 4.37. But since the difference was not significant, we chose the value 5 for this parameter.

Table 4.13 Tabu Search - Fitness values with the intensifyIterations parameter (phase III)

	Fitness				
intensifyIterations	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
3	0.304	0.637	0.872	1.079	1.245
5	0.313	0.643	0.870	1.068	1.230
7	0.374	0.650	0.867	1.071	1.229
9	0.379	0.660	0.878	1.083	1.246
11	0.399	0.656	0.880	1.082	1.246

Target	0.294	0.652	0.885	1.090	1.246

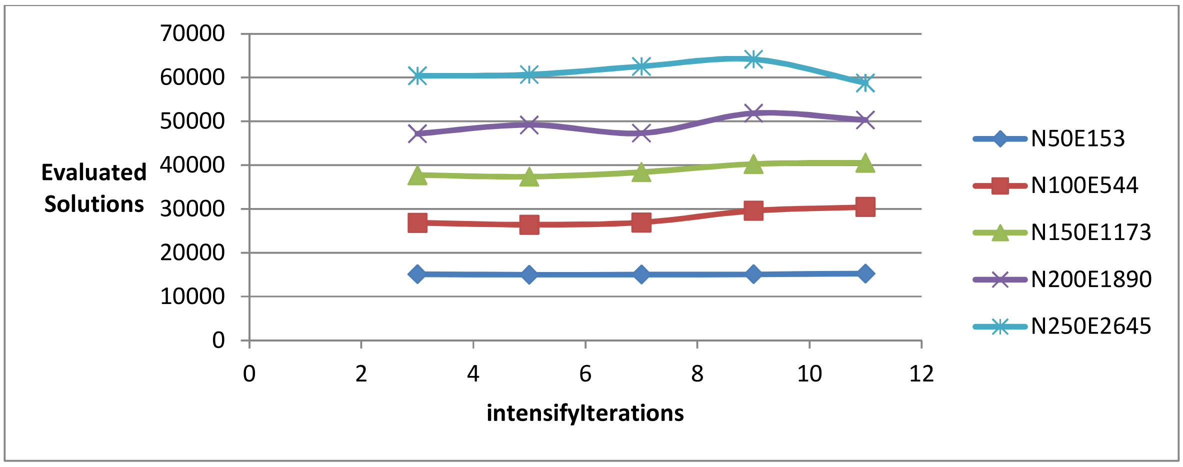


Figure 4.37 Tabu Search – Number of evaluated solutions with the intensifyIterations parameter (phase III)

The *duration* parameter has no significant effect on the quality of the produced layout as shown in the previous phases. However, increasing the value of this parameter to a certain limit would improve the performance of the drawing algorithm and consequently produce a smaller number of evaluated solutions. In this phase, we tested the value of *duration* with the following values: 5, 15, 25, 35, 45. Testing the algorithm with all these values produced graph layouts with quality at least as good as the target layout, as shown in Table 4.14. On the other hand, Figure 4.38 shows the number of accesses to the tabu list by the drawing algorithm. The higher the number of accesses to the tabu list, the higher the number of solutions to exclude from the searching process, and this consequently reduces the number of evaluated solutions. In the figure, it is indicated that the number of accesses to the tabu list increased between the values 5 and 15 and then became stable. Therefore, we selected the value 15 for the *duration* parameter.

Table 4.14 Tabu Search - Fitness values with the duration parameter (phase III)

	Fitness				
duration	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
5	0.313	0.643	0.870	1.068	1.230
15	0.324	0.645	0.866	1.070	1.230
25	0.328	0.644	0.866	1.071	1.229
35	0.327	0.645	0.867	1.070	1.232
45	0.327	0.645	0.867	1.070	1.232

Target	0.294	0.652	0.885	1.090	1.246

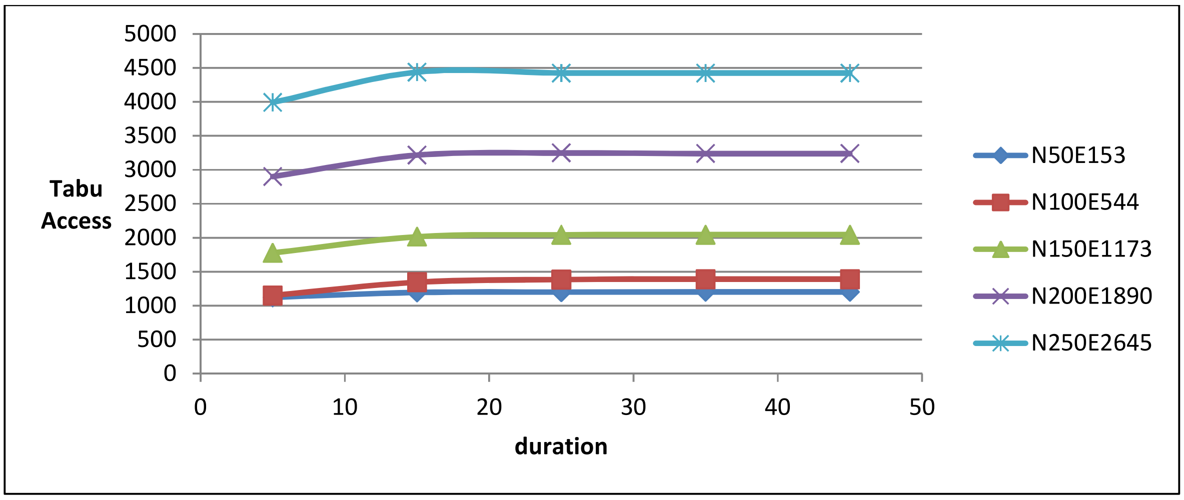


Figure 4.38 Tabu Search – Number of accesses to the tabu list with the duration parameter (phase III)

iv. Phase IV

We repeated the same tuning process that we performed on the tabu search drawing algorithm in phase III using the best initial square size as described in the hill climbing parameter tuning process. Table 4.15 and Figure 4.39 show the results produced by the tabu search drawing algorithm when we tested the *maxIterations* parameter with the values: 30, 35, 40, 45, 50. The main goal is to speed up the performance of the algorithm while producing graphs with good layouts. Therefore, we chose the value that best satisfies the target fitness values with the smallest number of iterations. The best value for *maxIterations* that satisfied this condition was 40.

Table 4.15 Tabu Search - Fitness values with the maxIterations parameter (phase IV)

	Fitness				
maxIterations	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
30	0.348	0.654	0.873	1.075	1.240
35	0.335	0.654	0.873	1.075	1.238
40	0.329	0.653	0.873	1.075	1.237
45	0.328	0.653	0.873	1.075	1.236
50	0.329	0.653	0.873	1.075	1.236

Target	0.294	0.652	0.885	1.090	1.246

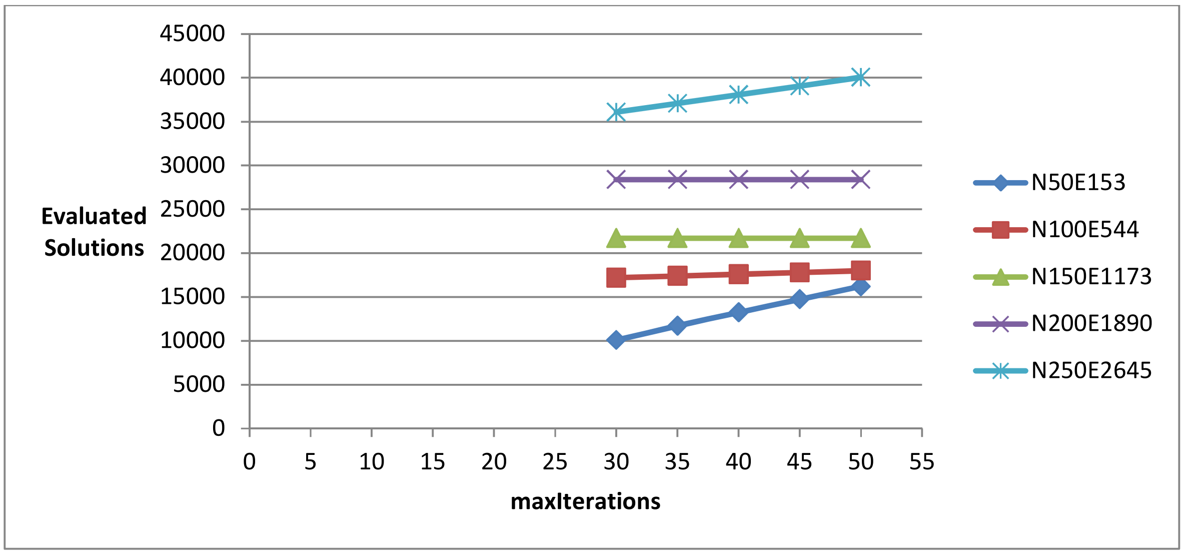


Figure 4.39 Tabu Search – Number of evaluated solutions with the maxIterations parameter (phase IV)

The *initialCutOff* parameter has been tested with the following values: 1, 2, 3, 4, 5. Table 4.16 shows that all the values (except the value 1) produced graph layouts with fitness values below or equal to the target (except for the first graphs set). On the other hand, Figure 4.40 demonstrates the number of evaluated solutions performed by the tabu search algorithm and indicates that when the algorithm uses the *initialCutOff* value 4, it generates the lowest number of solutions compared to the other values.

Table 4.16 Tabu Search - Fitness values with the initialCutOff parameter (phase IV)

	Fitness				
initialCutOff	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
1	0.892	1.080	1.261	1.479	1.648
2	0.329	0.653	0.873	1.075	1.237
3	0.324	0.653	0.873	1.077	1.238
4	0.316	0.652	0.877	1.079	1.236
5	0.322	0.660	0.876	1.079	1.241

Target	0.294	0.652	0.885	1.090	1.246

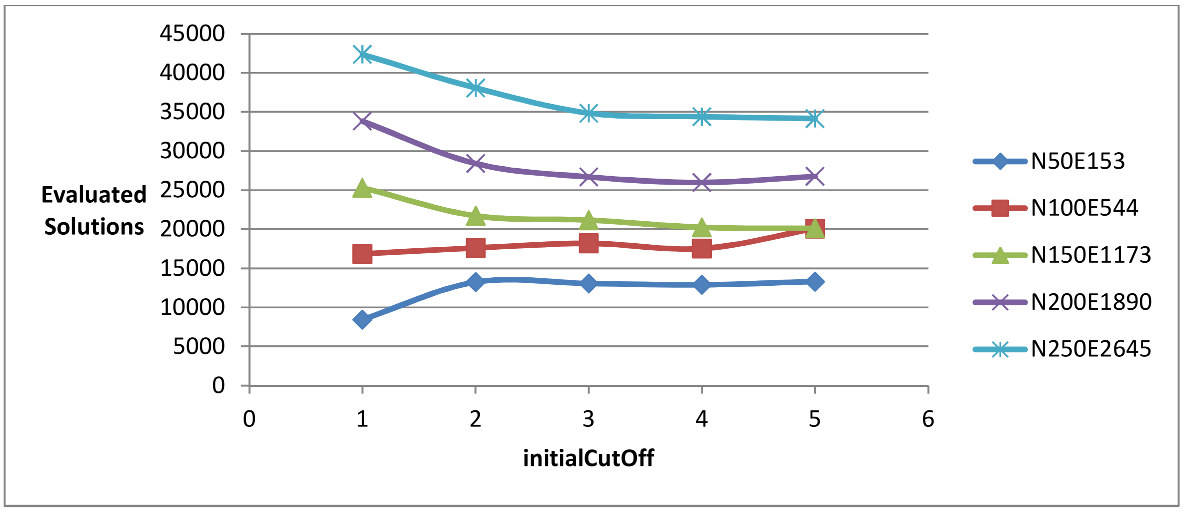


Figure 4.40 Tabu Search – Number of evaluated solutions with the initialCutOff parameter (phase IV)

As for the *intensifyCutOff* parameter, we tested it with the following values: 0.0025, 0.005, 0.0075, 0.01, 0.0125. Table 4.17 shows that all these values could give good layouts since all of them have reached fitness values less than or almost equal to the target fitness values (except for the first set of graphs). But in Figure 4.41, we realise that using any of these values would make no significant difference in the number of evaluated solutions performed by the algorithm with a slight advantage to the value 0.005 in most of the tested graph data sets. Therefore, we selected the value 0.005 for *intensifyCutOff*.

Table 4.17 Tabu Search - Fitness values with the intensifyCutOff parameter (phase IV)

	Fitness				
intensifyCutOff	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
0.0025	0.324	0.653	0.873	1.077	1.238
0.005	0.324	0.644	0.876	1.080	1.238
0.0075	0.320	0.645	0.874	1.077	1.239
0.01	0.321	0.647	0.872	1.080	1.236
0.0125	0.315	0.646	0.872	1.079	1.242
Target	0.294	0.652	0.885	1.090	1.246

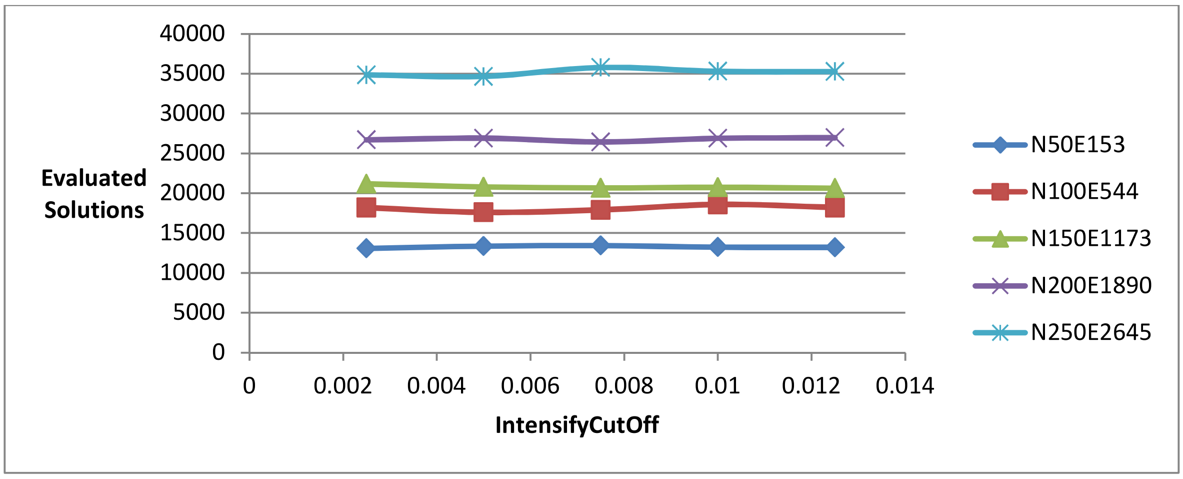


Figure 4.41 Tabu Search – Number of evaluated solutions with the intensifyCutOff parameter (phase IV)

The *intensifyIterations* parameter has been tested with five values 3, 5, 7, 9, 11. Figure 4.42 shows that the number of evaluated solutions increases as the value of this parameter increases. Picking the value 3 would be the best in terms of the number of evaluated solutions. However, according to Table 4.18, selecting this value would not produce good graph layouts for the first two sets of the graphs under test. The results in the table indicate that the fitness values in the first two sets of graphs (when *intensifyIterations* = 3) are far from the target fitness values. Therefore, we chose the next best value for *intensifyIterations* which equals to 5.

Table 4.18 Tabu Search - Fitness values with the intensifyIterations parameter (phase IV)

	Fitness				
intensifyIterations	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
3	0.367	0.660	0.877	1.077	1.240
5	0.324	0.644	0.876	1.080	1.238
7	0.310	0.652	0.881	1.080	1.240
9	0.335	0.655	0.877	1.080	1.245
11	0.399	0.670	0.878	1.082	1.247

Target	0.294	0.652	0.885	1.090	1.246

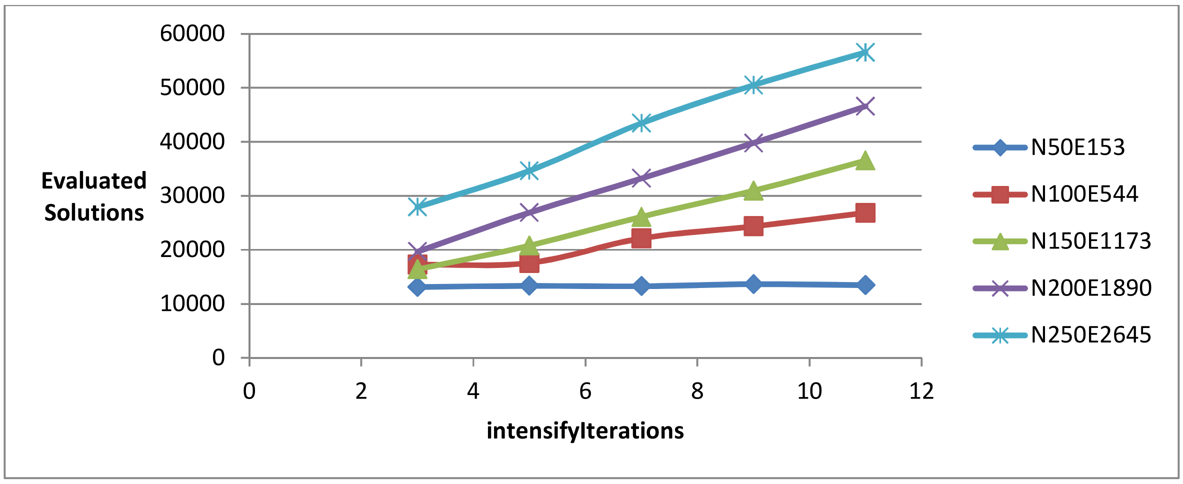


Figure 4.42 Tabu Search – Number of evaluated solutions with the intensifyIterations parameter (phase IV)

Last but not least, we tested the *duration* parameter with the following values: 5, 15, 25, 35, 45. Note that the search space in graph drawing is large, thus this parameter has no significant effect on the quality of the produced layouts as shown in Table 4.19, where all the tested values of this parameter have produced similar results. On the other hand, Figure 4.43 shows that this parameter has slightly affected the number of evaluated solutions performed by the tabu search drawing algorithm. The figure shows that there is a difference between the number of evaluated solutions when the value of *duration* is 5 and the rest of the values. Although the figure does not show that the difference is significant, but the difference could reach up to 7% (which reaches about 10,000 solutions or even more in some test cases where the number of nodes is very large). Thus, the value 5 is the most appropriate for the *duration* parameter.

Table 4.19 Tabu Aearch - Fitness values with the duration parameter (phase IV)

	Fitness				
duration	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
5	0.324	0.644	0.876	1.080	1.238
15	0.331	0.657	0.873	1.078	1.238
25	0.330	0.657	0.874	1.080	1.238
35	0.330	0.657	0.874	1.080	1.238
45	0.330	0.657	0.874	1.080	1.238

Target	0.294	0.652	0.885	1.090	1.246

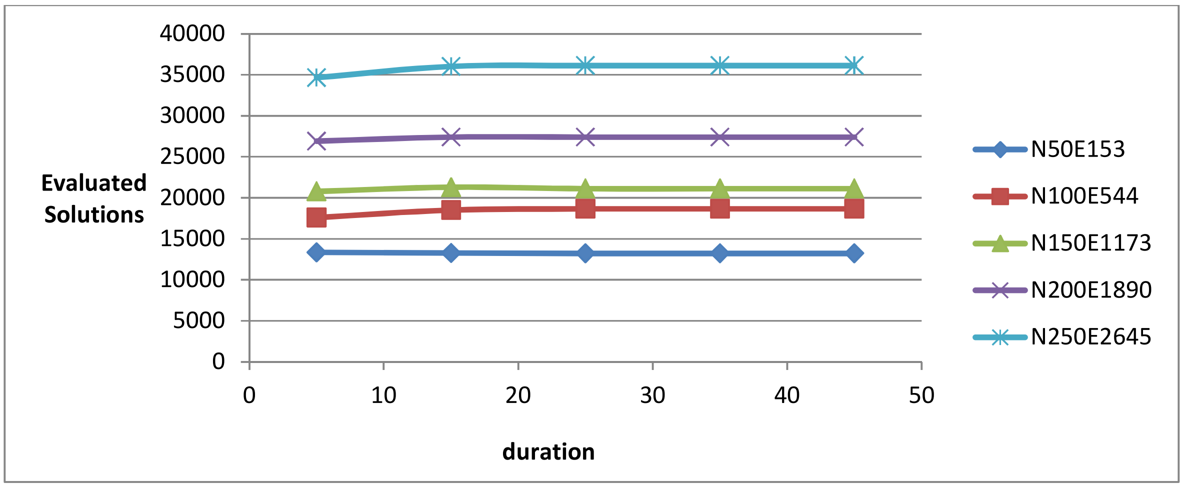


Figure 4.43 Tabu Search – Number of evaluated solutions with the duration parameter (phase IV)

Now as we have tuned all the parameters of the three methods, we list the value of each parameter which will be used in our coming experiments:

i. Hill Climbing Parameters

initialSquareSize = 512

squareReduction = 4

ii. Simulated Annealing Parameters

maxIterations = 45

iterPerTemp = 15

initialTemp = 0.75

coolDown = 0.8

iii. Tabu Search Parameters

maxIterations = 40

initialCutOff = 4

intensifyCutOff = 0.005

intensifyIterations = 5

duration = 5

4.6 Summary

This chapter described the basic neighbourhood search-based graph drawing algorithms for hill climbing and simulated annealing, followed by our tabu search-based approach for drawing general graph layouts with straight lines that have multiple aesthetic criteria which are used in a weighted fitness function to measure the quality of the graph layout. Each criterion had a different range of values. Hence, a normalisation process for the values to a unified range was described.

This chapter also demonstrated how the three drawing algorithms had used the same local search space. They also shared the same procedure for tuning the values of their parameters by performing exploratory tests on a wide range of values for each parameter in order to select a robust set of initial values. Then a systematic incremental procedure was applied for each single parameter at a time while fixing the values of the rest of the parameters.

Hill climbing, simulated annealing, and tabu search graph drawing algorithms were described including their pseudo codes and a complete description of their parameters. A detailed clarification of the parameter tuning process for each parameter was demonstrated including figures and tables that showed the effect of each parameter on the quality of the layouts and the efficiency of the drawing algorithms.

In the next chapter, we show the experimental results of a comprehensive comparison between the three neighbourhood search-based methods according to the quality of the generated layouts and the efficiency of the algorithms.

Chapter 5 Experimental Results of Comparing Hill Climbing, Simulated Annealing, and Tabu Search

This chapter demonstrates the experimental results of applying the three graph drawing algorithms described in the previous chapter: hill climbing, simulated annealing, and our graph drawing version of tabu search on random graph datasets and real world graph datasets. It also shows our analysis and conclusions to the results.

5.1 Introduction

Our research question in this experiment was: 'Does our tabu search graph drawing algorithm perform better than the hill climbing and simulated annealing approaches?' To answer this question we had to implement and evaluate our method against the two commonly used alternative neighbourhood search-based methods for graph drawing. Three types of evaluations were conducted:

i. Finding the best layout that can be achieved (i.e. minimising the value of the fitness function);

ii. How long it took to draw a graph to a particular level of quality;

iii. How good the quality of the graph was after a fixed optimisation time (number of evaluated solutions).

These allow us to examine different possible use cases for the graph layout: firstly, generating the best possible layout; secondly speed to draw an acceptable layout; and thirdly how good the graph layout can be if there is a fixed time available to produce it.

The programming language used in our implementation is Java (version 1.7.0; Java HotSpot™ 64-Bit Server VM 21.0-b17 on Windows 7). The experiments were performed using Lenovo Thinkpad T430, Intel® Core™ i7-3520M CPU processor with 2.90 GHz frequency and 8 GB RAM.

We generated random graph datasets in two categories. The graphs of the first category have the same number of nodes but with different densities (i.e. different number of edges), whereas the graphs of the second category have a different number of nodes with varying values of densities.

The random graph generator is based on the Erdos-Renyi model (Erdos & Rényi 1960; Daudin et al. 2008). It generated randomly connected graphs. The parameters to the generator were the number of nodes and the density of the graph. Random locations for the nodes were generated based on the size of the window where the graph is displayed. Then, the generator chose random nodes as the end points of the edges. All random values were generated using the random method in Java. Self-sourcing edges and multiple edges between the same pair of nodes were not allowed. Finally, the graph generator tested the connectivity of the generated graph. Only connected graphs were accepted. In our implementation for the random graph generator, we added an option which allows the user to randomly change the layout of the generated connected graph.

There were 80 random graphs in the first category split into 4 groups of 20 test cases each. All the graphs in this category had 150 nodes, randomly positioned. Each group had a differing number of edges so that the density varied. The graphs in each group had same number of nodes and edges. See Table 5.1 for the characteristics of the graphs in the first category. Note that the density of the graph is computed using the same formula described in the previous chapter (Section 4.3.2).

Table 5.1 Characteristics of the graphs in the 1st category

Graph Set	Nodes	Edges	Density
1A	150	558	0.05
2A	150	1117	0.1
3A	150	1676	0.15
4A	150	2235	0.2

The second category also had 80 random graphs, again divided into 4 groups. The number of nodes for a group varied, increasing in steps of 50. The value of the density was chosen for each group to avoid too dense graphs so that we could generate graphs that were easily visualised. A similar random process used to generate graphs in the first category was applied to this category. See Table 5.2 for the characteristics of the graphs in the second category.

Table 5.2 Characteristics of the graphs in the 2nd category

Graph Set	Nodes	Edges	Density
1B	50	159	0.13
2B	100	569	0.115
3B	150	1173	0.105
4B	200	1990	0.1

The initial layout of nodes for each graph was random. We applied our tabu search-based approach along with hill climbing and simulated annealing approaches to the graphs. Tabu search and hill climbing approaches are deterministic methods which are not influenced by chance. The characteristic of this type of method is that the output is determined when the set of input elements and properties in the model has been specified. Both methods use the same initial input layout and there is no randomness in their implementation. On the other hand, simulated annealing is a stochastic method which includes an element of randomness in the neighbourhood searching process. Therefore, this approach has been tested on each individual graph for 30 different runs. Then we find the median of the results for the 30 different runs to compare with the results of the tabu search and hill climbing approaches. Note that, we modelled the neighbourhood transition probability of simulated annealing in a similar way to the model described in Davidson & Harel (1996). In the following section, we describe the three phases of the experiment along with the analysis of the results.

5.2 Experiments on Random Graph Datasets

To make a comprehensive comparison between the methods, we divided our experiment into three phases. Firstly, in phase I, we focus on the overall performance for each method regardless of how long it takes to execute to get the best possible graph layout that can be generated by that method. Secondly, in phase II, we study the speed of each algorithm when it runs to draw a graph for a particular level of quality. Thirdly, in phase III, we investigate the quality of the drawn layouts after a fixed predefined execution time.

5.2.1 Phase I

We applied the methods on the graphs of the two categories described in Section 5.1. The methods executed on the 20 test cases in each group of the two categories, and then the average fitness function value and the average number of evaluated solutions were computed for each group in each method. Note that in simulated annealing, the average of medians was computed for the 30 runs of each test case. In this phase, the hill climbing approach was executed until it found the best solution that can be reached (i.e. a solution that cannot be further improved). Whereas, the simulated annealing and tabu search approaches were more flexible in how they reach a good solution, and hence we ran them using the values of the parameters discussed earlier in the previous chapter.

The following figures show bar charts of the results obtained from phase I. Figure 5.1 and Figure 5.2 show the difference between the three methods in terms of the quality of the produced layouts (fitness value) when applied on each category of graphs, whereas Figure 5.3 and Figure 5.4 show the difference according to the performance efficiency (number of evaluated solutions).

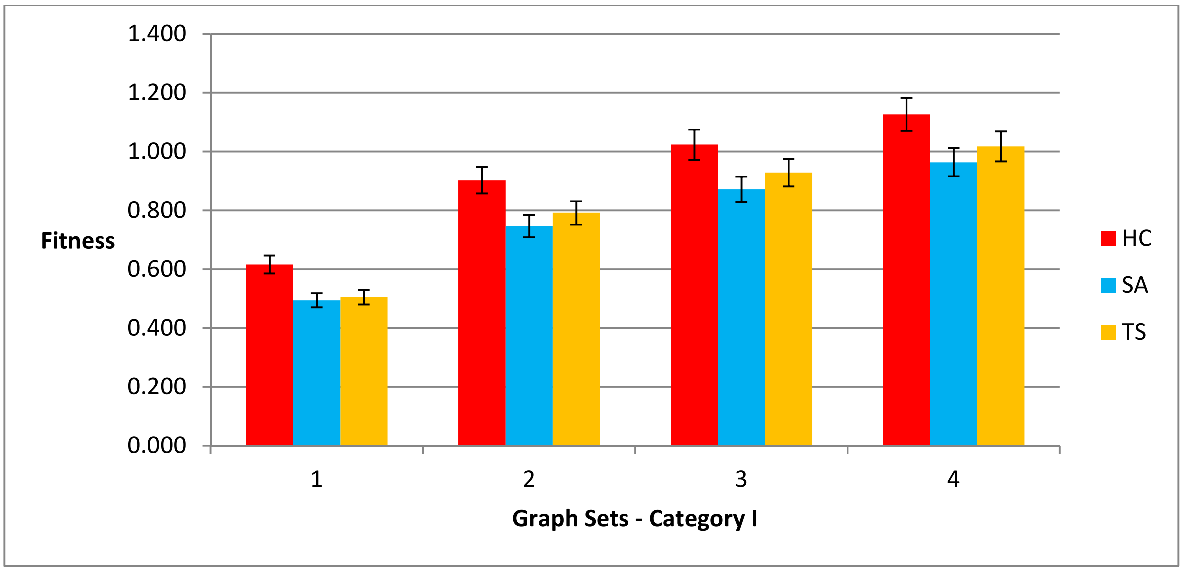


Figure 5.1 Bar chart with 95% confidence interval of the fitness function obtained by HC, SA, TS when applied on the graphs of the 1st category (phase I)

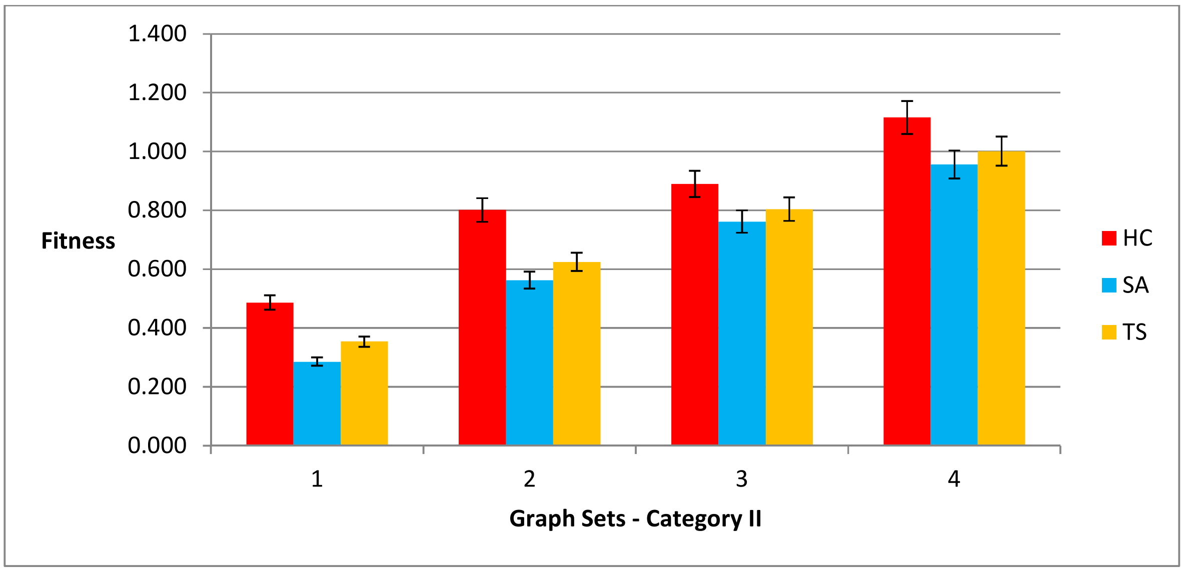


Figure 5.2 Bar chart with 95% confidence interval of the fitness function obtained by HC, SA, TS when applied on the graphs of the 2nd category (phase I)

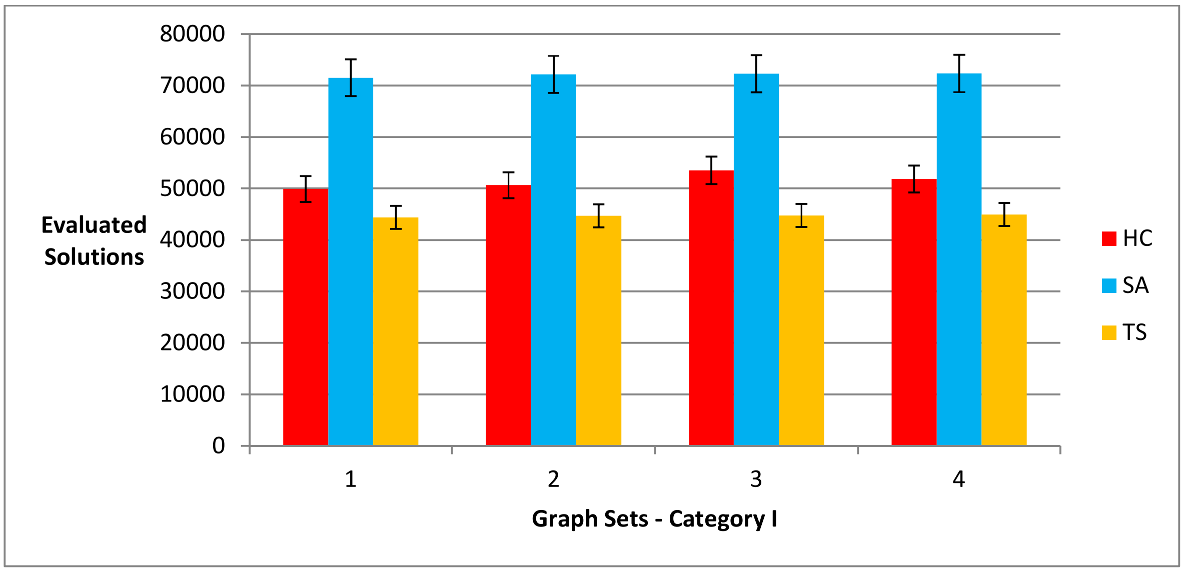


Figure 5.3 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of the 1st category (phase I)

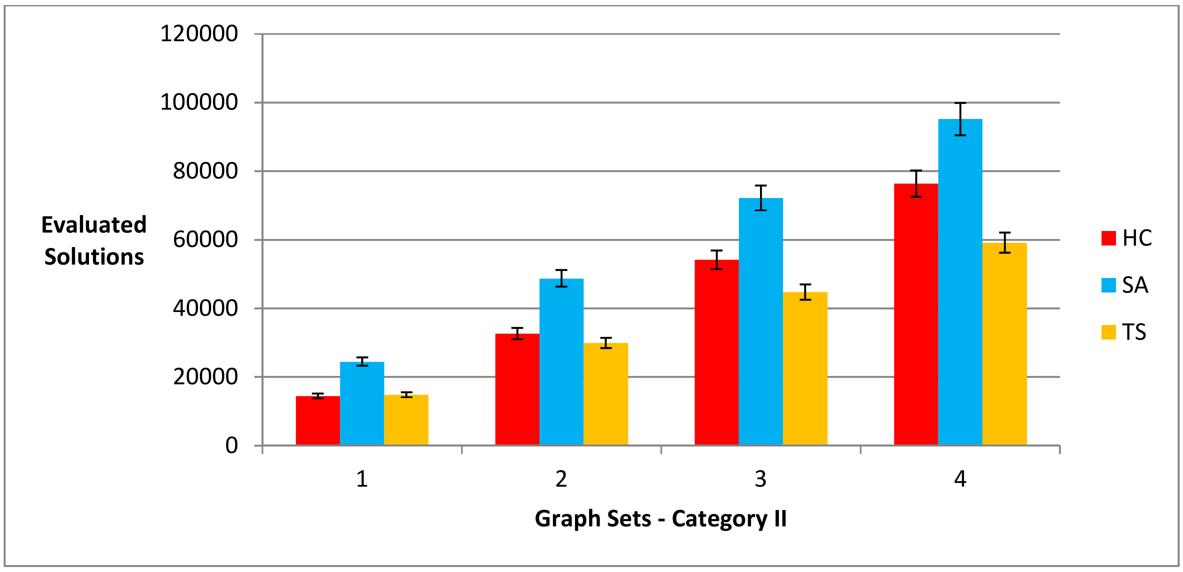


Figure 5.4 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of the 2nd category (phase I)

Figure 5.5 and Figure 5.6 show the execution time (in seconds) when the methods were applied on the data of the first and second categories respectively. The figures demonstrate how lengthy the layout process was with simulated annealing compared to the other two methods. Our proposed tabu search-based method, on the other hand, shows a slightly faster execution time against hill climbing.

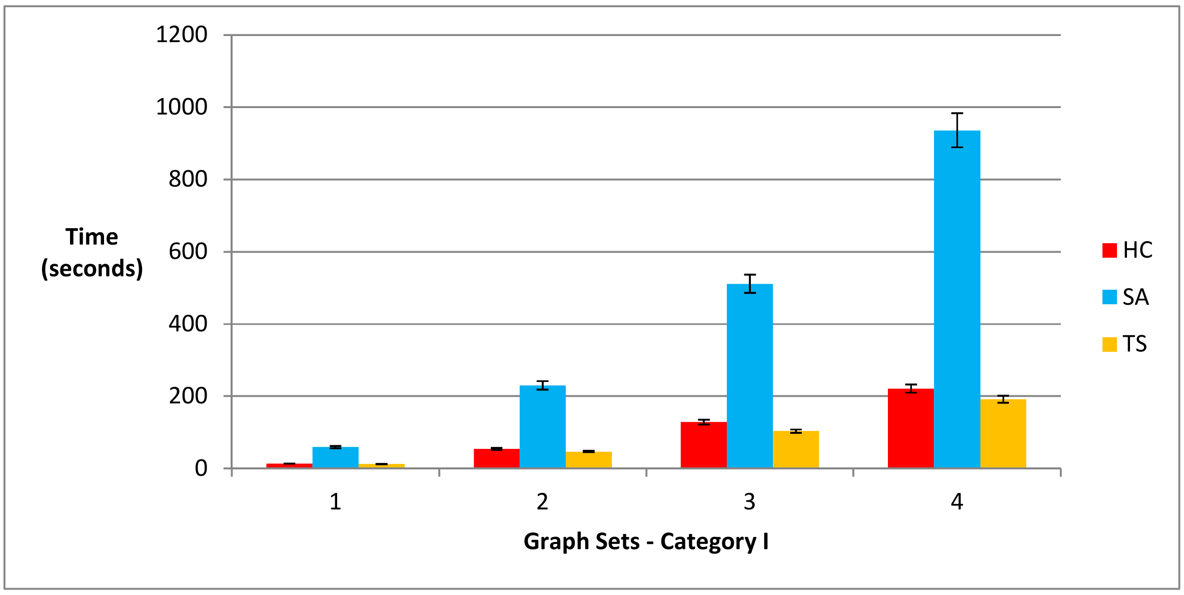


Figure 5.5 Bar chart with 95% confidence interval of the execution time (in seconds) obtained by HC, SA, TS when applied on the graphs of the 1st category (phase I)

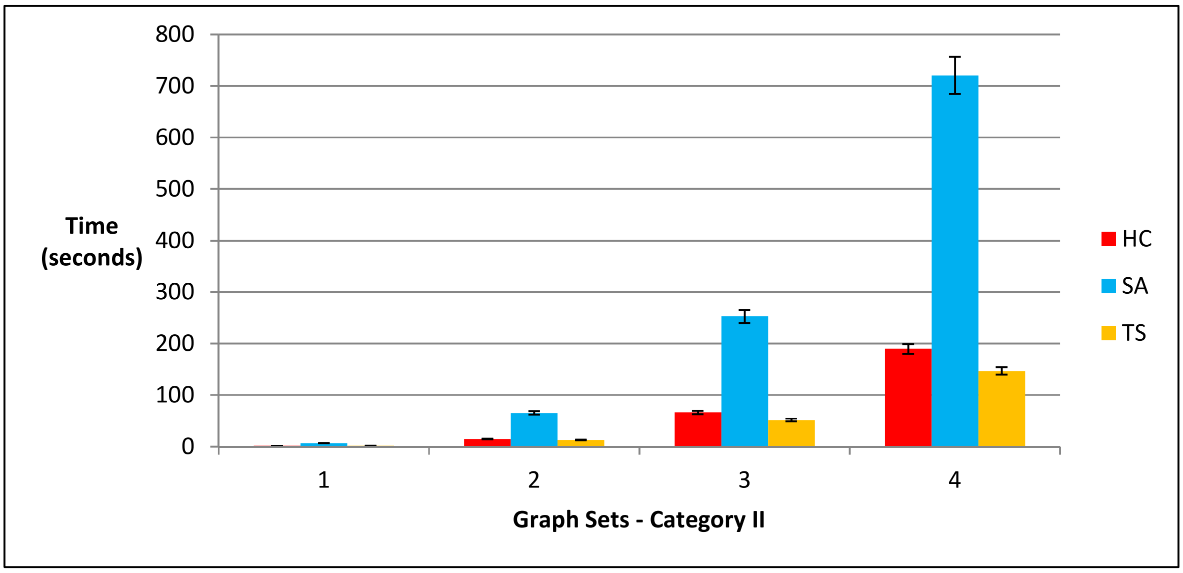


Figure 5.6 Bar chart with 95% confidence interval of the execution time (in seconds) obtained by HC, SA, TS when applied on the graphs of the 2nd category (phase I)

In Figure 5.7 and Figure 5.8, we merge the results obtained from applying the three methods on both categories (category I and category II graph datasets) to show respectively the average overall fitness value and the average number of evaluated solutions produced by the three methods. On the other hand, Table 5.3 and Table 5.4 demonstrate the statistical analysis of the fitness values for the graph layouts produced by the three methods when applied on the graph datasets of the first and the second categories together along with the number of evaluated solutions obtained by each method. You can refer to Section 5.2.4 for a complete description of the conducted statistical test and for the interpretation of the p-value column listed in the tables.

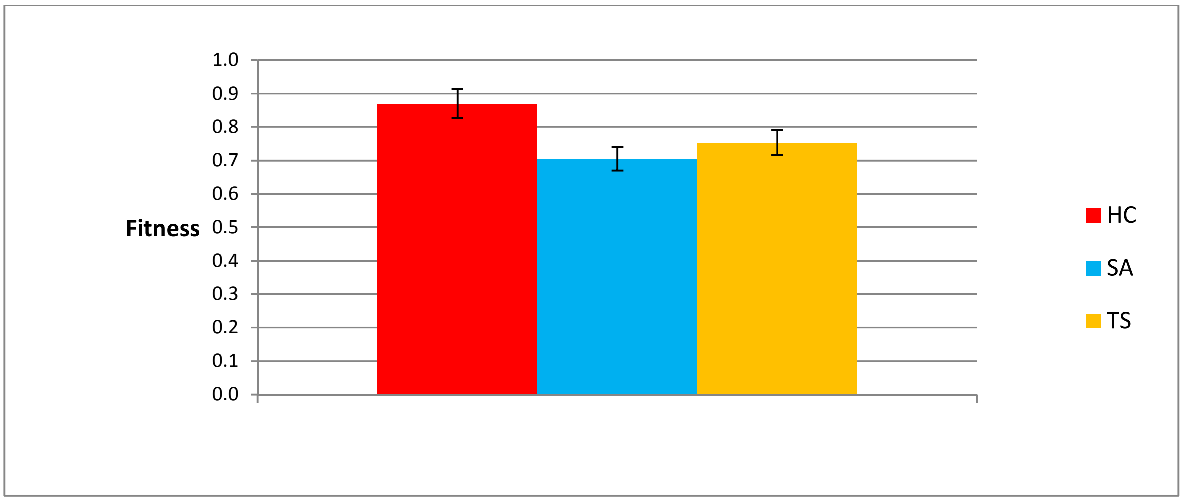


Figure 5.7 Bar chart with 95% confidence interval of the average overall fitness function obtained by HC, SA, TS when applied on the graphs of both categories (phase I)

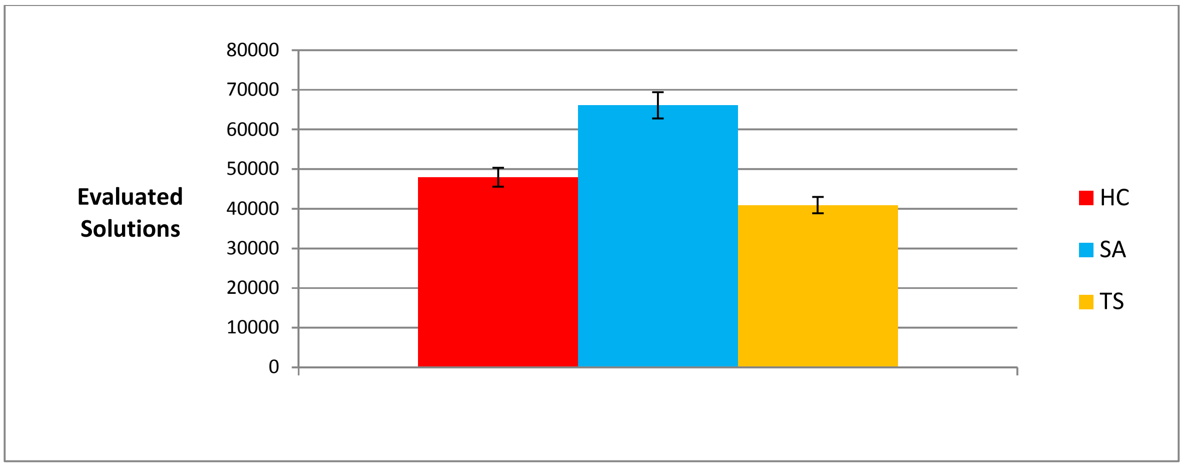


Figure 5.8 Bar chart with 95% confidence interval of the average overall number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of both categories (phase I)

Table 5.3 Statistical analysis of the fitness function for HC, SA, TS when applied on the graphs of both categories (phase I)

	Fitness												
	Hill Climbing				Simulated Annealing				Tabu Search				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1A	0.616	0.607	0.828	0.502	0.494	0.494	0.503	0.484	0.505	0.504	0.558	0.421	4.11E-07
2A	0.902	0.875	1.211	0.791	0.746	0.746	0.754	0.728	0.791	0.784	0.869	0.728	5.33E-09
3A	1.023	0.978	1.309	0.916	0.871	0.871	0.884	0.860	0.928	0.922	1.061	0.889	2.06E-09
4A	1.126	1.092	1.387	0.987	0.963	0.964	0.974	0.955	1.017	1.013	1.154	0.944	5.33E-09
1B	0.486	0.465	0.827	0.361	0.286	0.286	0.297	0.272	0.354	0.332	0.618	0.280	1.25E-08
2B	0.801	0.743	1.210	0.614	0.563	0.562	0.587	0.543	0.625	0.612	0.794	0.551	6.52E-09
3B	0.890	0.863	1.249	0.800	0.762	0.761	0.777	0.754	0.805	0.801	0.948	0.730	3.56E-08
4B	1.116	1.076	1.504	0.979	0.956	0.958	0.968	0.918	1.001	0.995	1.072	0.942	6.52E-09
Overall	0.870	0.837	1.191	0.744	0.705	0.705	0.718	0.689	0.753	0.745	0.884	0.685	

Table 5.4 Statistical analysis of number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of both categories (phase I)

	Evaluated Solutions												
	Hill Climbing				Simulated Annealing				Tabu Search				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1A	49867	50071	56715	40577	71480	71473	71582	71326	44391	44393	44656	44047	2.64E-08
2A	50623	50132	60846	39727	72149	72139	72266	72059	44688	44670	45020	44381	1.25E-08
3A	53516	52571	65036	42458	72287	72277	72438	72186	44765	44783	45112	44368	1.25E-08
4A	51838	51640	68429	39193	72343	72342	72540	72162	44941	44939	45357	44382	2.64E-08
1B	14523	14206	18779	11801	24485	24489	24634	24387	14870	14883	15010	14619	1.38E-07
2B	32643	32661	44387	25746	48740	48733	48936	48638	29918	29959	30287	29454	8.76E-08
3B	54128	51864	71345	42643	72203	72208	72340	72068	44741	44764	45086	44243	5.33E-09
4B	76351	76891	93479	58574	95171	95163	95327	95079	59182	59152	59775	58818	5.33E-09
Overall	47936	47504	59877	37590	66107	66103	66258	65988	40937	40943	41288	40539	

The results presented in Figure 5.1, Figure 5.2, Figure 5.7, and Table 5.3 show that simulated annealing produces the best graph layouts compared to the other two methods. It has a slight advantage over tabu search in the quality of the graph layout, but both are considerably better than hill climbing. On the other hand, simulated annealing evaluates a larger number of solutions in order to get those good layouts. Figure 5.3, Figure 5.4, Figure 5.8, and Table 5.4 show that tabu search outperforms the other two methods in terms of performance efficiency (number of evaluated solutions). The figures in Appendix A (A.1 and A.2) are samples of the layouts drawn by the three algorithms when applied on the graph datasets described in Table 5.1 and Table 5.2 respectively.

5.2.2 Phase II

In phase II, we investigated the performance of the approaches rather than the quality of the produced layouts. The following process was performed on the graphs of the two categories, described in Section 5.1, to test which method has the lowest number of evaluated solutions to reach a particular value of a fitness function (a particular level of a layout quality):

1. We ran the hill climbing method on the graphs until no improvements could be made to the value of the fitness function. We started with hill climbing, in particular, because in phase I, it produced graph layouts with the worst quality compared to the other two methods. Therefore, simulated annealing and tabu search could easily produce graph layouts with good quality as the one produced by hill climbing.

2. We ran simulated annealing and tabu search methods until they reached an equal or better fitness function value compared to the one found by the hill climbing drawing algorithm.

3. We measured the number of evaluated solutions for each method.

Figure 5.9 and Figure 5.10 show the number of evaluated solutions obtained by the three methods when they are applied on the graphs of the first category and the second category respectively. Whereas Figure 5.11 and Table 5.5 describe, visually and statistically, the average overall number of evaluated solutions obtained from phase II when the three methods are applied on the graphs of the two categories together.

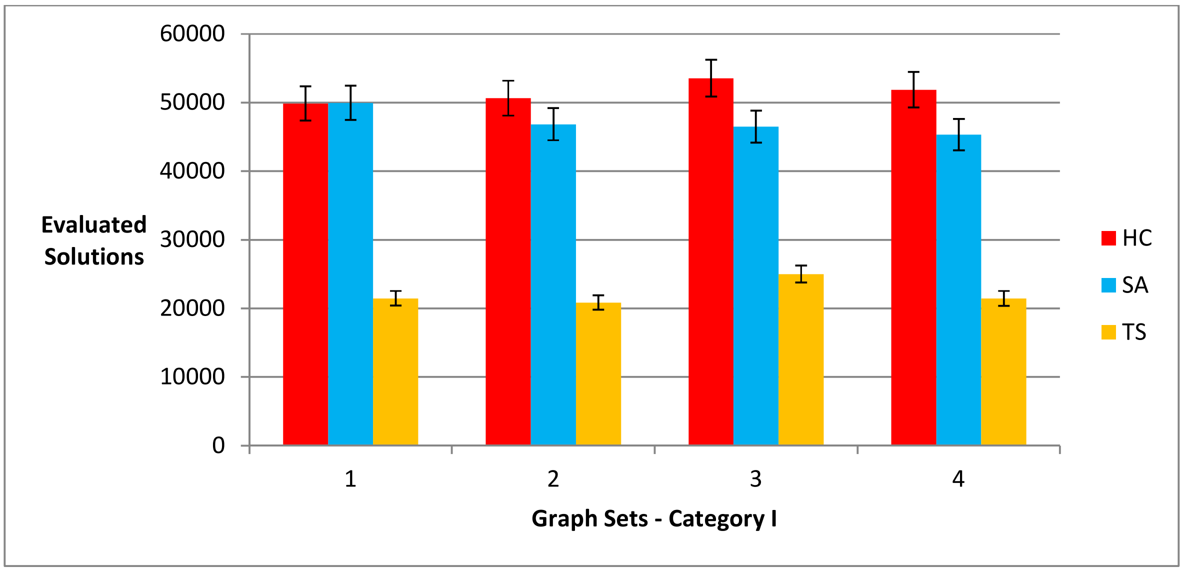


Figure 5.9 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of the 1st category (phase II)

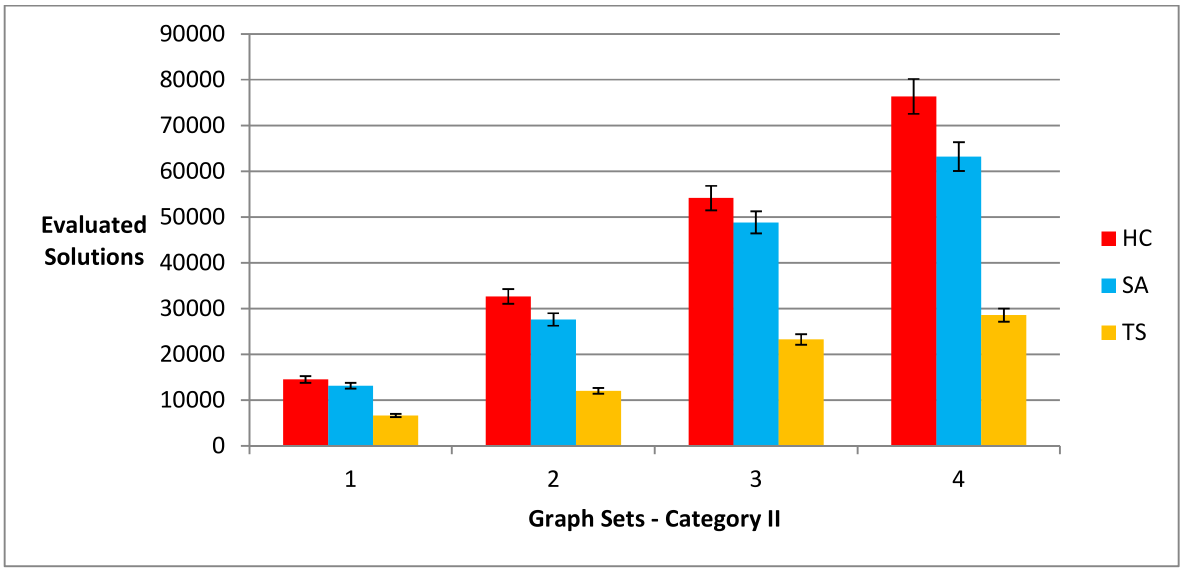


Figure 5.10 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of the 2nd category (phase II)

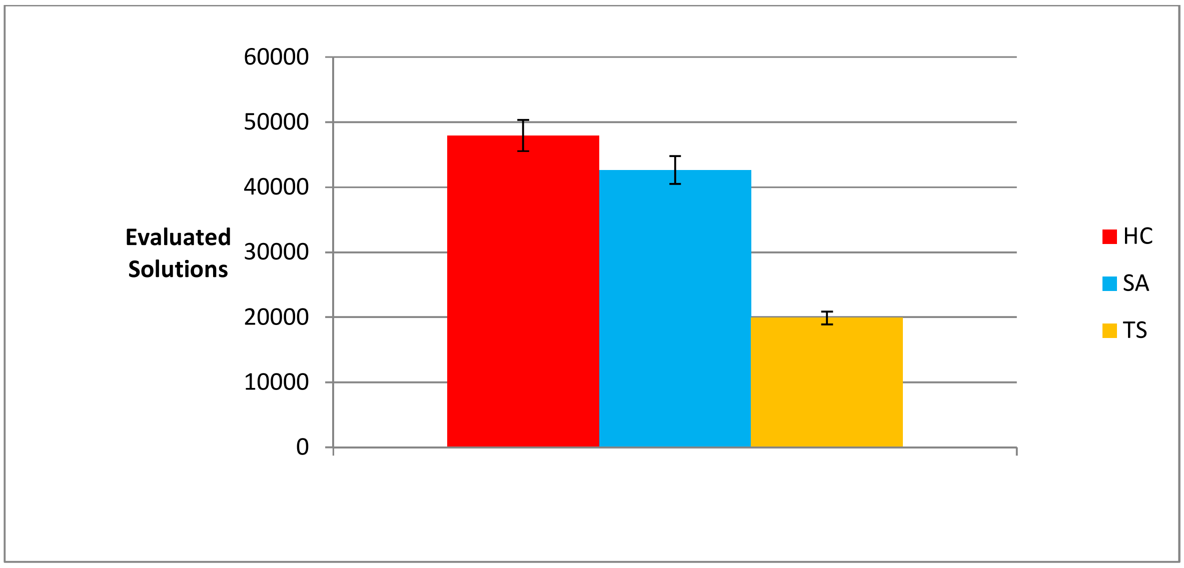


Figure 5.11 Bar chart with 95% confidence interval of the average overall number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of the two categories together (phase II)

Table 5.5 Statistical analysis of the average overall number of evaluated solutions obtained by HC, SA, TS when applied on the graphs of the two categories together (phase II)

	Evaluated Solutions												
	HC				**SA**				**TS**				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1A	49867	50070	56715	40577	49929	50073	70029	37054	21468	23010	28765	12633	2.51E-07
2A	50622	50131	60846	39727	46822	46122	67079	32120	20851	23205	29366	2272	8.76E-08
3A	53516	52570	65036	42458	46478	47600	60463	32053	25007	27351	41053	6086	2.64E-08
4A	51837	51640	68429	39193	45321	45549	61512	32090	21450	25254	29789	2299	8.76E-08
1B	14523	14205	18779	11801	13136	12388	19220	7822	6665	6142	11819	2690	7.16E-07
2B	32643	32661	44387	25746	27602	26903	41822	16976	12009	11855	22726	2677	5.06E-08
3B	54127	51863	71345	42643	48811	49601	58243	31749	23266	23461	44243	5984	4.80E-07
4B	76351	76891	93479	58574	63208	63516	87893	41797	28551	30759	39755	1790	2.64E-08
Overall	**47936**	**47504**	**59877**	**37589**	**42663**	**42719**	**58282**	**28957**	**19908**	**21379**	**30939**	**4553**	**< 2.2e-16**

According to the results shown in Figure 5.9, Figure 5.10, Figure 5.11, and Table 5.5, we conclude that our tabu search method generates graph layouts of good quality with a very limited number of evaluated solutions compared to hill climbing and simulated annealing. This difference is significant since the p-values for all graph sets are smaller than our significance level as shown in the last column of the tables according to the Friedman test that will be described in Section 5.2.4.

5.2.3 Phase III

In phase III, we investigated the quality of the layout produced by the drawing algorithms rather than the performance. The following process was performed to test which method produces the graph layouts with the best quality (smallest value of fitness function) when the three methods perform the same number of evaluated solutions:

1. We ran the tabu search method on the graphs for a predefined number of iterations (maxIterations = 40 as described in Chapter 4, Section 4.5.2). The number of evaluated solutions is computed and saved. We started with the tabu search in particular because in phase I, it generated the lowest number of evaluated solutions.

2. We ran hill climbing and simulated annealing methods until they perform the same number of evaluated solutions performed by the tabu search drawing algorithm.

3. We measured the value of the fitness function produced by the drawing algorithms in each of the steps above.

Figure 5.12 and Figure 5.13 show the values of the fitness function obtained by the three drawing algorithms when they are applied on the graphs of the first category and the second category respectively. Whereas Figure 5.14 and Table 5.6 describe, visually and statistically, the average overall fitness function values obtained from phase III when the three methods are applied on the graphs of the two categories together.

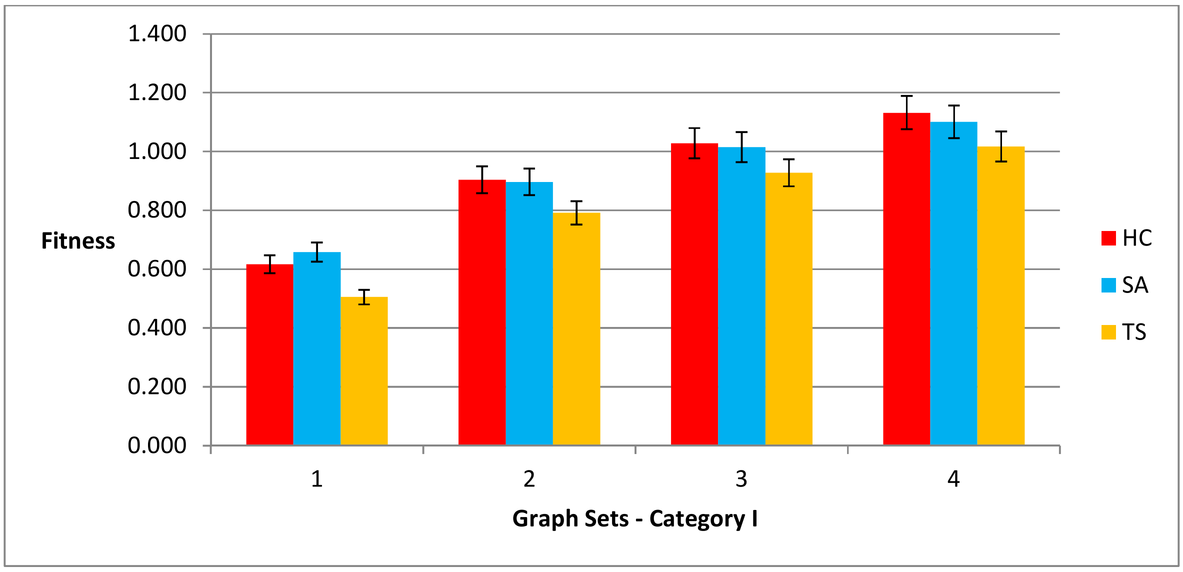


Figure 5.12 Bar chart with 95% confidence interval of the fitness function values obtained by HC, SA, TS when applied on the graphs of the 1st category (phase III)

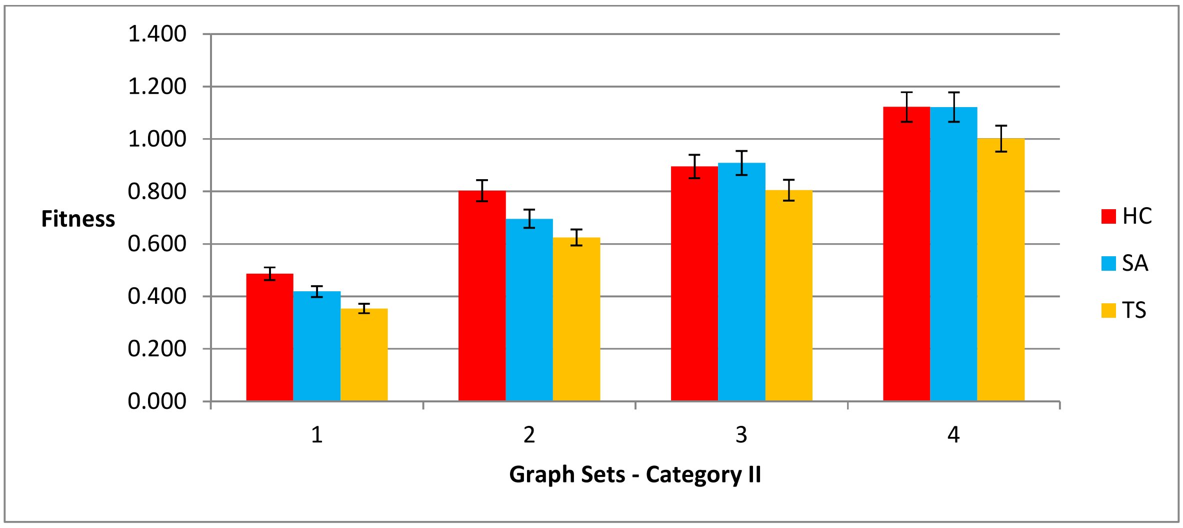


Figure 5.13 Bar chart with 95% confidence interval of the fitness function values obtained by HC, SA, TS when applied on the graphs of the 2nd category (phase III)

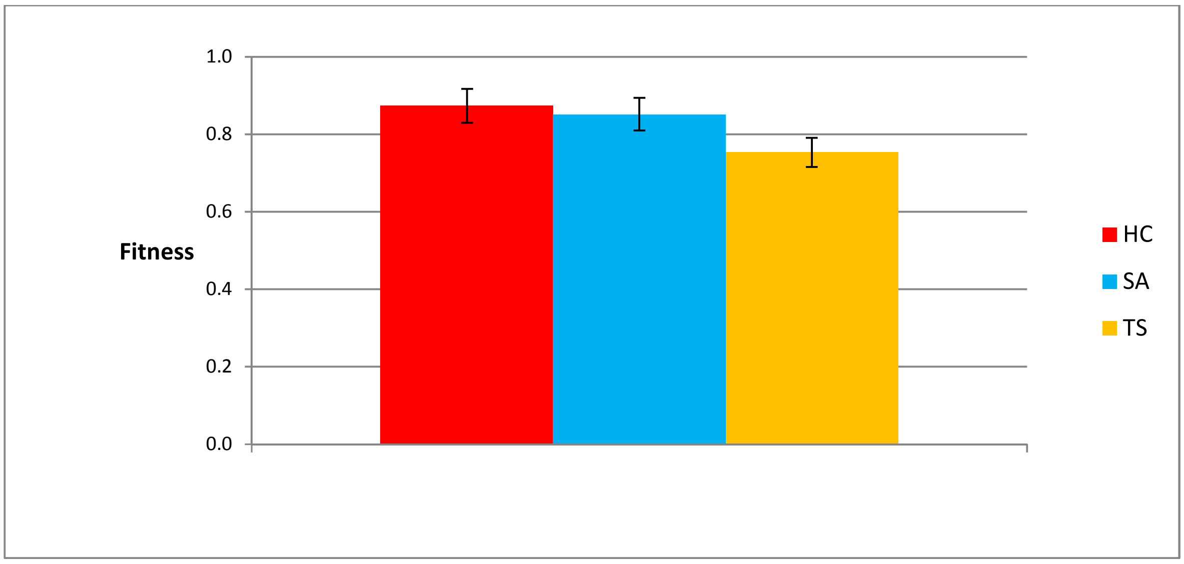


Figure 5.14 Bar chart with 95% confidence interval of the average overall fitness function values obtained by HC, SA, TS when applied on the graphs of the two categories together (phase III)

Table 5.6 Statistical analysis of the average overall fitness function values obtained by HC, SA, TS when applied on the graphs of the two categories together (phase III)

	Fitness												
	HC				**SA**				**TS**				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1A	0.617	0.609	0.828	0.502	0.658	0.659	0.668	0.646	0.505	0.504	0.558	0.421	2.64E-08
2A	0.904	0.877	1.211	0.792	0.897	0.897	0.907	0.886	0.791	0.784	0.869	0.728	1.38E-07
3A	1.028	0.989	1.309	0.925	1.015	1.015	1.033	1.002	0.928	0.922	1.061	0.889	9.66E-07
4A	1.132	1.098	1.390	0.988	1.101	1.100	1.123	1.090	1.017	1.013	1.154	0.944	1.30E-06
1B	0.487	0.465	0.827	0.361	0.419	0.421	0.438	0.390	0.354	0.332	0.618	0.280	9.80E-07
2B	0.803	0.746	1.210	0.616	0.696	0.696	0.713	0.683	0.625	0.612	0.794	0.551	9.66E-07
3B	0.895	0.872	1.249	0.803	0.908	0.909	0.921	0.895	0.805	0.801	0.948	0.730	4.80E-07
4B	1.122	1.082	1.517	0.987	1.121	1.123	1.138	1.102	1.001	0.995	1.072	0.942	1.36E-07
Overall	**0.873**	**0.842**	**1.193**	**0.747**	**0.852**	**0.853**	**0.868**	**0.837**	**0.753**	**0.745**	**0.884**	**0.685**	**< 2.2e-16**

We conclude from the results presented in Figure 5.12, Figure 5.13, Figure 5.14, and Table 5.6 that our tabu search approach draws graph layouts with better quality (or similar quality in the worst case) compared to hill climbing and simulated annealing when they evaluate the same number of solutions. The Friedman statistical significance test was applied on the results and the p-values in the tables show that there is a significant difference in the layouts between the three methods.

Figure 5.15 and Figure 5.16 show two different examples of random graph layouts drawn by hill climbing, simulated annealing, and our tabu search approach.

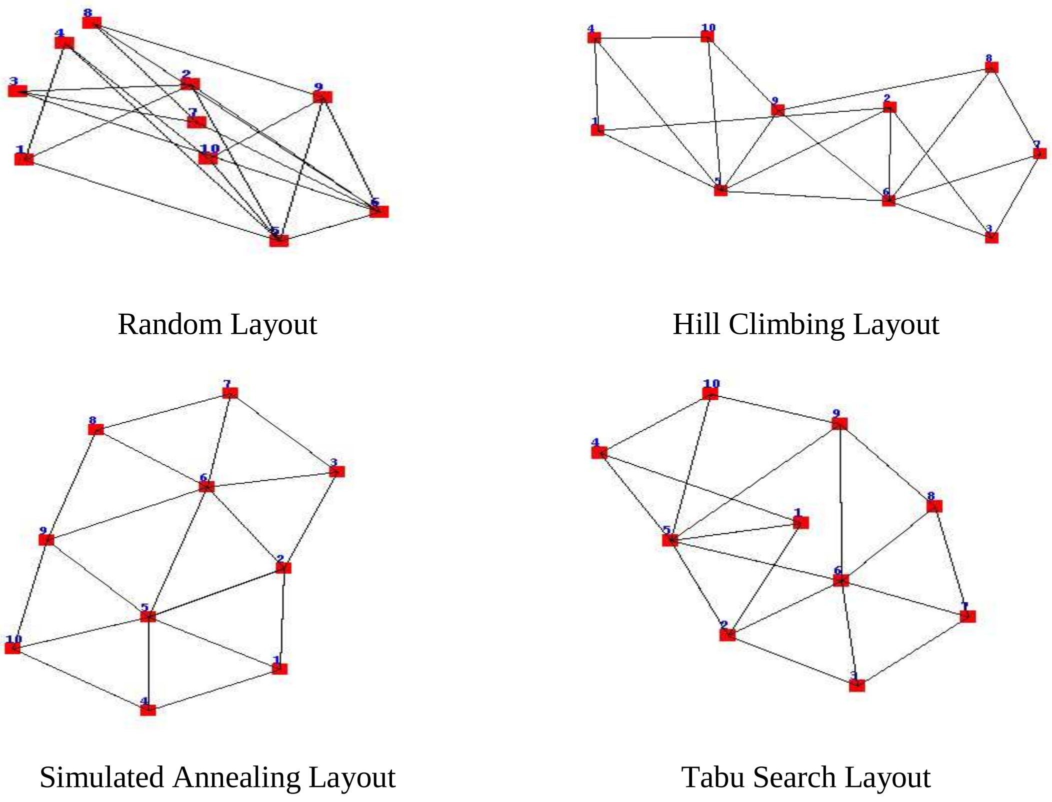


Figure 5.15 Example of connected graph layout with 10 nodes and 19 edges drawn within the canvas of our visualization tool by the three methods: HC, SA, TS

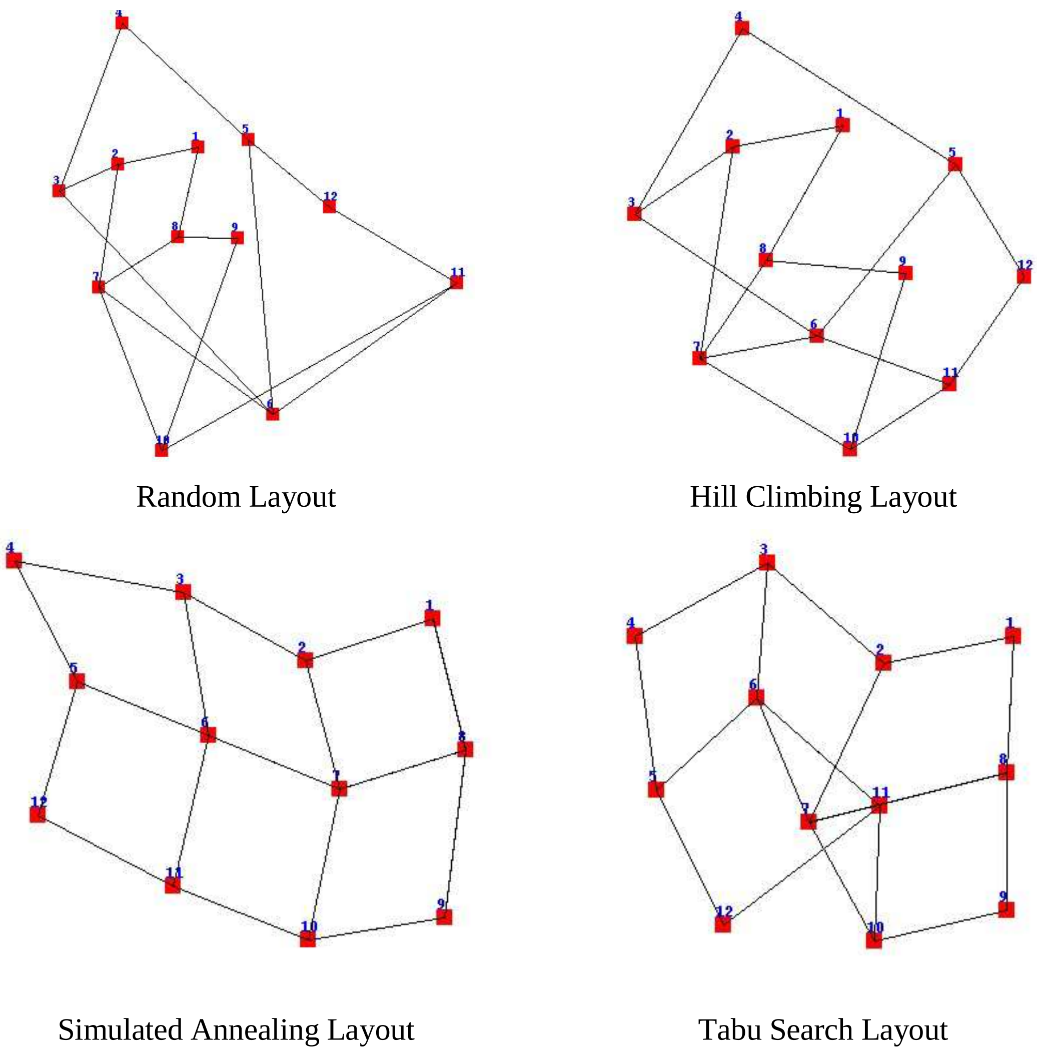


Figure 5.16 Example of connected graph layout with 12 nodes and 17 edges drawn within the canvas of our visualization tool by the three methods: HC, SA, TS

5.2.4 Statistical Tests

In order to test the effect of randomness in generating the initial graph layouts used in comparing the methods, we performed a statistical significance test on the results generated from the three phases. Note that, we applied a statistical significance test on phase I for the fitness values of the graph layouts generated by the three methods to conclude which method draws the best layout without fixing a specific number of evaluated solutions performed by each method. To demonstrate that there is a statistical significant difference between the three methods, we first applied the Friedman test (Upton & Cook 2014) which is a non-parametric test for testing the differences between several samples. This test requires no prior knowledge of the distribution of data. We could have applied ANOVA test if our population was normally distributed, but when we applied Shapiro-Wilk normality test (Shapiro & Wilk 1965) on our randomly generated datasets, we got p-values less than the significance level that equals to 0.05. Thus, the null hypothesis of Shapiro-Wilk's test that the population is normally distributed was rejected.

We ran the three methods on 20 randomly generated test cases, based on Erdos-Renyi model, for each group of graphs in the first and second categories. Note that, in simulated annealing, we calculated the median of 30 runs for each test case instead of finding the mean (median is more reliable in avoiding outliers) and consequently we got 20 medians (since we find the mean of 30 medians for each test case). Then we compared them with the results of the means computed by hill climbing and tabu search using the Friedman test with a significance level of a value 0.05. The null hypothesis for the Friedman test states that there are no differences between the results of the methods. If the probability is low (i.e. less than the selected significance level) the null hypothesis is rejected and it can be concluded that at least two methods are significantly different from each other. In all the tests, as shown in Table 5.3, Table 5.4, Table 5.5, and Table 5.6, we got p-values smaller than 0.05 which means we can reject the null hypothesis and hence we conclude that there is a significant difference between the three methods.

The Friedman test allowed us to conclude that there is a significant difference between the methods, but it does not show how each method differs from the other. Therefore, a post-hoc test for multiple comparisons between the methods was conducted using the Wilcoxon signed-rank test (Wilcoxon 1945) with Bonferroni correction (Dunn 1961; Holm 1979). The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test that can be used as an alternative to the paired student's t-test since our population is not normally distributed. Bonferroni correction, on the other hand, is a simple method that allows pairwise comparisons and is easy to apply. Despite the importance of using the Bonferroni method for the multiple comparison post-hoc correction, it can be considered conservative if there are a large number of tests and/or the test statistics are positively correlated (Perneger 1998). Note that all the statistical tests were conducted using the R statistical package i386 (version 3.1.1).

In the Bonferroni correction, we lower the significance level value to 0.01 in an attempt to prevent data from incorrectly appearing to be statistically significant and to increase the accuracy of results. When you are performing many independent or dependent statistical tests at the same time, this multiple comparison post-hoc correction is used (Bland & Altman 1995).

While a p-value of a statistical significance test can indicate the existence of a significant difference, but it does not show the size of that difference. Effect size is a simple way of quantifying the difference between the results of two methods. Here, we measure it by the standardized difference between two means (i.e., difference of means divided by the standard deviation). Cohen (1992) classified effect sizes as small (= 0.2), medium (= 0.5), and large (= 0.8). See Table 5.7, Table 5.8, Table 5.9, and Table 5.10 for the effect sizes and p-values in phase I; Table 5.11 and Table 5.12 for the effect sizes and p-values in phase II; and Table 5.13 and Table 5.14 for the effect sizes and p values in phase III.

The p-values of the Bonferroni post-hoc test shown in Table 5.7 and Table 5.8 conclude that there is a significant difference between the layouts drawn by simulated annealing and the other two methods except in the first graph dataset of the first category. The effect size of fitness between simulated annealing and hill climbing is always large, whereas the effect size increases from medium to large as the graph size increases when simulated annealing is compared against tabu search. On the other hand, Table 5.9 and Table 5.10 show that the tabu search outperforms the other two methods in terms of performance efficiency as the number of nodes increases except for small graphs as shown in the first and the second graph datasets in the second category in Table 5.10 (i.e., the effect size increases from small to medium then large when tabu search is compared against hill climbing, and it is always large when compared against simulated annealing).

Table 5.7 Effect size and p-values for the fitness function values after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 1st category (phase I)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	1.7e-10	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.0442	0	0.9987	0	1.0374	0	1.1315	0
TS	p	1.2e-08	0.2400	3.4e-06	1.5e-07	0.0001	4.4e-11	0.0002	1.2e-06
	effect	0.8999	0.2954	0.7372	0.9724	0.6957	0.8648	0.7848	0.7805

Table 5.8 Effect size and p-values for the fitness function values after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 2nd category (phase I)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	4.4e-11	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.1044	0	1.0903	0	0.7858	0	0.8629	0
TS	p	4.1e-06	2.3e-06	1.0e-05	1.5e-06	1.2e-05	1.0e-05	1.7e-05	2.3e-06
	effect	0.7310	0.5909	0.8397	0.7571	0.5297	0.6295	0.6400	0.9621

Table 5.9 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 1st category (phase I)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	2.0e-07	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.6794	0	1.6181	0	1.4865	0	1.4912	0
TS	p	0.0002	2.0e-07	2.6e-05	2.0e-07	8.7e-06	4.4e-11	0.0002	4.4e-11
	effect	1.0772	1.8704	0.9268	1.8705	1.1500	1.8694	0.7738	1.8677

Table 5.10 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 2nd category (phase I)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	4.4e-11	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.8559	0	1.6627	0	1.3728	0	1.1804	0
TS	p	0.1400	4.4e-11	0.0280	4.4e-11	1.2e-07	4.4e-11	1.2e-07	4.4e-11
	effect	-0.1824	1.8629	0.4961	1.8668	0.9798	1.8682	1.3144	1.8725

In Table 5.11 and Table 5.12 we see that tabu search outperforms hill climbing and simulated annealing in drawing graph layouts with similar fitness values using a lower number of evaluated solutions as the p-values in the tables show that there is a statistical significant difference between the tabu search and the other two methods along with very large effect sizes. On the other hand, there is no statistically significant difference in the number of evaluated solutions between simulated annealing and hill climbing when applied on graphs with a small number of nodes. However, Table 5.12 shows that there is a significant difference between the two methods as the number of nodes increases with medium effect sizes.

Table 5.11 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 1st category (phase II)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	1.0000	*	0.1500	*	0.0260	*	0.0340	*
	effect	0.0045	0	-0.2736	0	-0.5751	0	-0.4570	0
TS	p	4.4e-11	4.4e-11	4.4e-11	4.4e-11	4.4e-11	5.2e-10	4.4e-11	4.4e-11
	effect	1.7633	1.4981	1.5434	1.3164	1.5074	1.2786	1.4538	1.3258

Table 5.12 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 2nd category (phase II)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	0.1700	*	0.0430	*	0.2200	*	0.0009	*
	effect	0.3465	0	0.4984	0	0.3879	0	0.6480	0
TS	p	8.7e-11	6.1e-09	4.4e-11	2.9e-09	8.7e-11	8.3e-10	4.4e-11	4.4e-11
	effect	1.4860	1.2484	1.5087	1.2869	1.4767	1.4578	1.5524	1.3110

Table 5.13 and Table 5.14 show that the tabu search always draws graph layouts with better quality compared to hill climbing and simulated annealing using the same number of evaluated solutions with medium to large effect sizes when compared against hill climbing, and very large effect sizes when compared against simulated annealing. On the other hand, there is no significant difference between the qualities of the graph layouts drawn by hill climbing and simulated annealing when they are applied on graphs using the same number of evaluated solutions.

Table 5.13 Effect size and p-values for the fitness function values after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 1st category (phase III)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	0.0300	*	0.1500	*	0.2400	*	0.9200	*
	effect	-0.3991	0	0.0525	0	0.1075	0	0.2562	0
TS	p	1.2e-08	4.4e-11	2.3e-06	4.4e-11	4.0e-05	1.2e-07	1.0e-04	1.2e-07
	effect	0.9060	1.6279	0.7494	1.5637	0.7368	1.4149	0.8186	1.2066

Table 5.14 Effect size and p-values for the fitness function values after conducting the Bonferroni test on HC, SA, TS when applied on the graphs of the 2nd category (phase III)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		HC	SA	HC	SA	HC	SA	HC	SA
SA	p	0.0430	*	0.5500	*	0.0180	*	0.1500	*
	effect	0.4500	0	0.5625	0	-0.0928	0	0.0048	0
TS	p	4.1e-06	1.2e-05	1.0e-05	2.8e-06	6.0e-06	1.2e-07	9.9e-07	4.4e-11
	effect	0.7319	0.6455	0.8482	0.9581	0.5628	1.3612	0.6625	1.6436

In the next section, we explore the performance of the methods when applied to real world datasets sourced from the Internet.

5.3 Experiments on Real World Graph Datasets

After performing several experiments on random graphs, we tested our system on real world graph datasets to demonstrate that we can reproduce similar results in a real world setting. We selected 10 different datasets from different sources as shown in Table 5.15 that also indicates the number of nodes, the number of edges, and the density of each graph. The graphs have different sizes with different densities. The initial layout of the nodes in each graph was generated randomly. Hill climbing and tabu search have run once on the same initial layout whereas simulated annealing has run 30 times on that random initial layout, as we previously did. Then we calculated the median for each of the 30 runs which was used in comparison with the results of the other two methods. We tested the methods according to phases I, II, and III described in Section 5.2.

Table 5.15 Real world graph datasets characteristics and sources

Graph	Nodes	Edges	Density	Source	Description
1	34	78	0.139	(Zachary 1977)	A social network of friendships between 34 members of a karate club at a US university in the 1970s
2	62	159	0.084	(Lusseau et al. 2003)	An undirected social network of frequent associations between 62 dolphins in a community living off Doubtful Sound, New Zealand
3	105	441	0.081	(Krebs n.d.)	Books about US politics sold by the online bookseller Amazon.com. Edges represent the frequent co-purchasing of books by the same buyers, as indicated by the ‘customers who bought this book also bought these other books’ feature on Amazon
4	112	425	0.068	(Newman 2006)	The network of common adjective and noun adjacencies for the novel ‘David Copperfield’ by Charles Dickens
5	115	613	0.094	(Girvan & Newman 2002)	The network of American football games between Division IA colleges during regular season Fall 2000
6	128	2075	0.255	(Melián & Bascompte 2004)	A network contains the carbon exchanges in the cypress wetlands of South Florida during the wet season
7	198	2742	0.141	(Gleiser & Danon 2003)	List of edges of the network of Jazz musicians

8	277	1918	0.05	(Choe et al. 2004)	C. elegans global network of 277 neurons, and the spatial positions of the neurons as two-dimensional coordinates
9	297	2148	0.049	(White et al. 1986)	Neural network of the nematode C. Elegans
10	332	2126	0.039	(Batagelj & Mrvar 2006)	Undirected weighted graph for US Air flights

The results of the experiments are shown in the following figures. We start with Figure 5.17 and Figure 5.18 that illustrate the results of applying the three methods on the real data graphs described in Table 5.15 according to phase I. The figures assert the conclusion formulated in Section 5.2.1 stipulating that simulated annealing draws the best graph layouts compared to hill climbing and tabu search, but it is the worst in terms of efficiency.

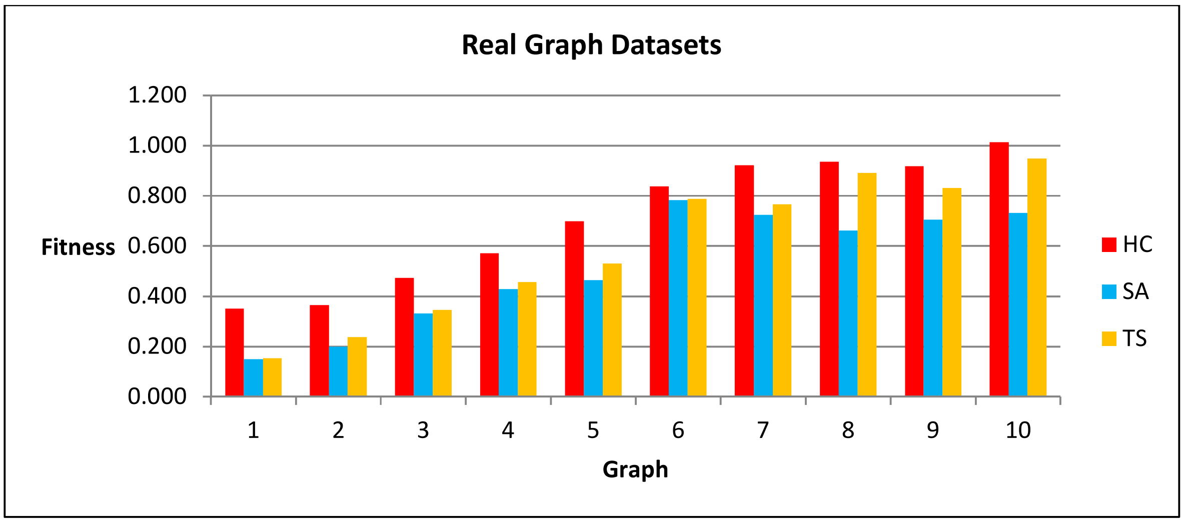


Figure 5.17 Bar chart of the fitness function values obtained by HC, SA, TS when applied on the graph datasets in Table 5.15 (phase I)

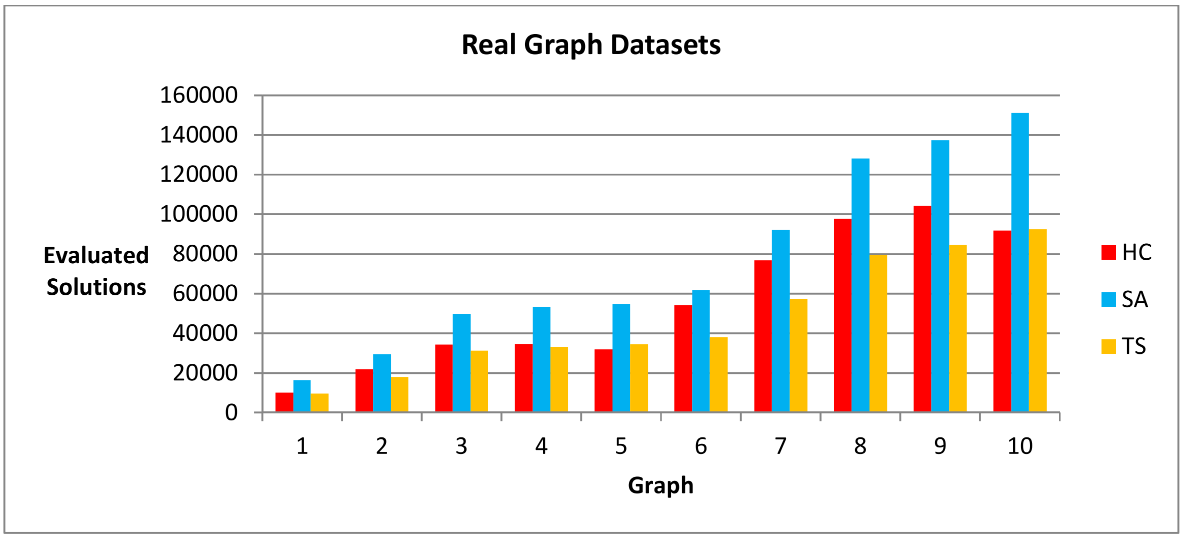


Figure 5.18 Bar chart of the number of evaluated solutions obtained by HC, SA, TS when applied on the graph datasets in Table 5.15 (phase I)

Figure 5.19 represents the number of evaluated solutions performed by each method when testing the methods on the real world graphs described in Table 5.15 according to phase II. The figure shows that our tabu search drawing algorithm outperforms the other two approaches as the size of the graph increases which supports the conclusion we had in Section 5.2.2 when the methods were applied on random graphs.

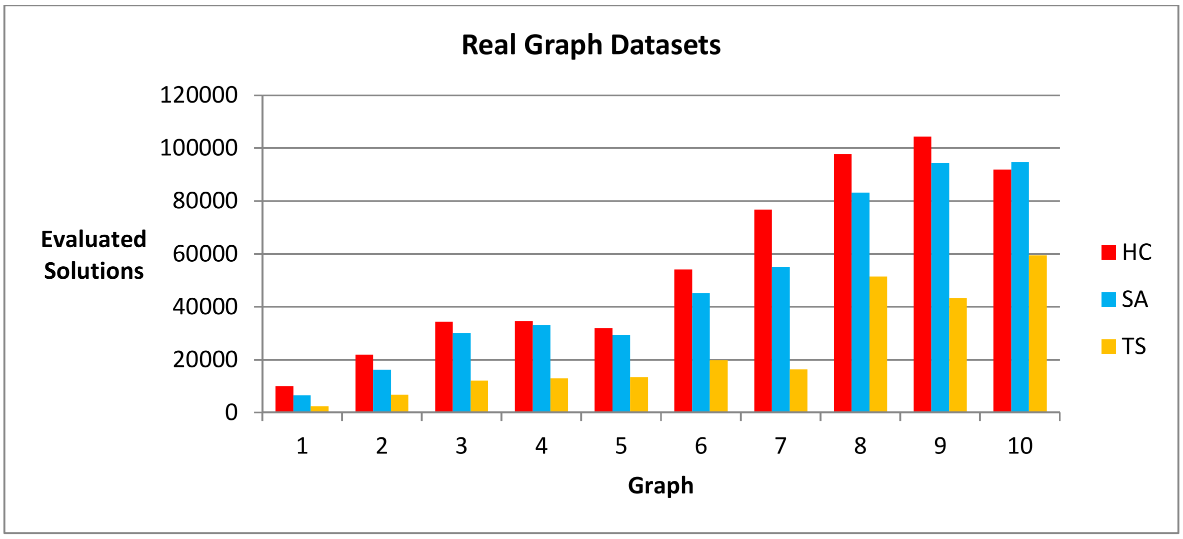


Figure 5.19 Bar chart of the number of evaluated solutions obtained by HC, SA, TS when applied on the graph datasets in Table 5.15 (phase II)

In Figure 5.20, the values of the fitness function obtained by each method are demonstrated following the experiment described in phase III when applied on the same set of data. The figure shows that the tabu search approach can reproduce the same behaviour described in Section 5.2.3 on the real data setting (i.e. tabu search approach produces graph layouts with better quality compared to hill climbing and simulated annealing when they evaluate the same number of solutions).

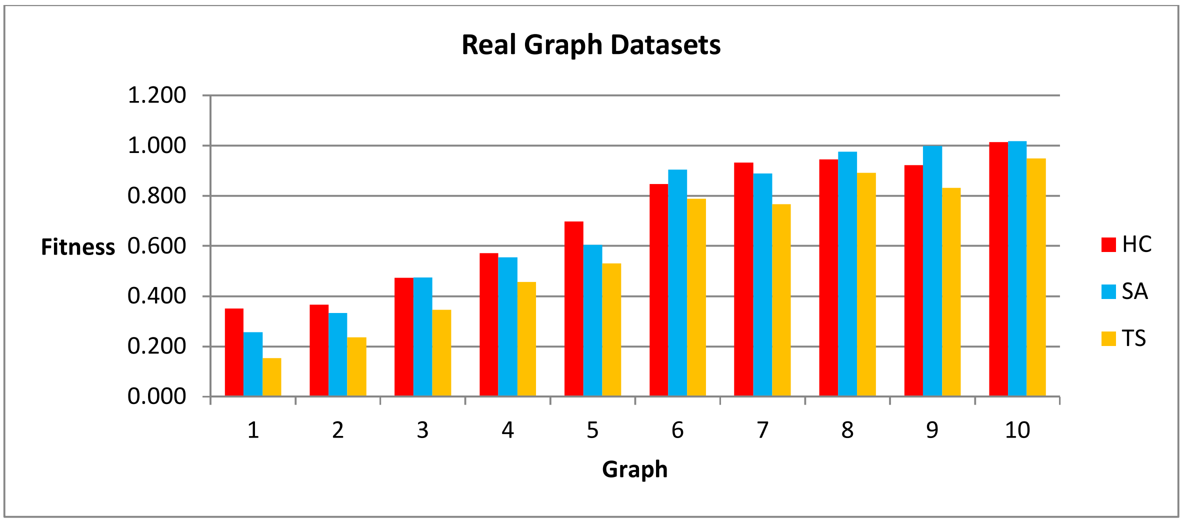
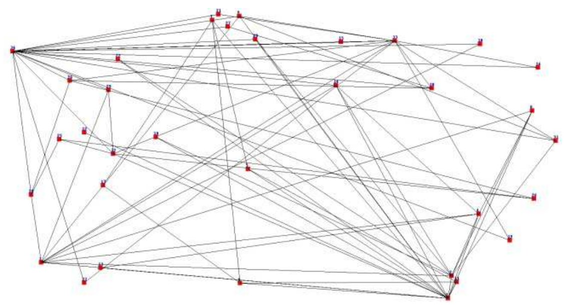
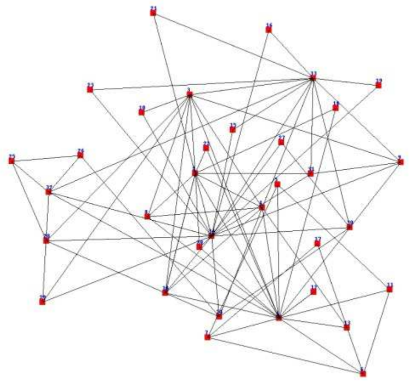
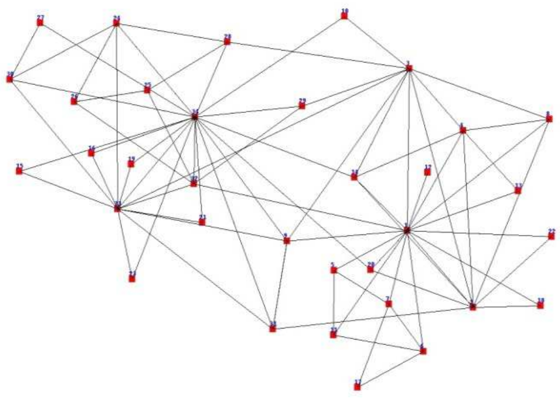


Figure 5.20 Bar chart of the fitness function values obtained by HC, SA, TS when applied on the graph datasets in Table 5.15 (phase III)

Figure 5.21 and Figure 5.22 are examples of the layouts produced by hill climbing, simulated annealing, and tabu search when applied to graph 1 and graph 2 in the list of real world datasets described in Table 5.15.


Random Layout


Hill Climbing Layout


Simulated Annealing Layout

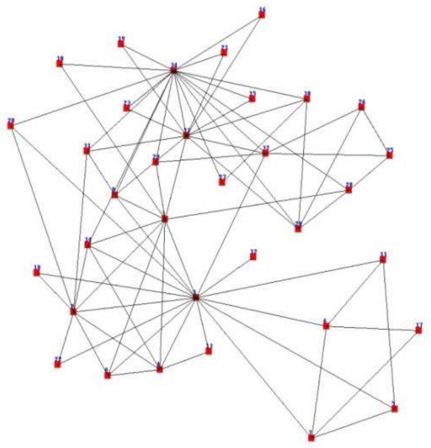

Tabu Search Layout

Figure 5.21 Layout of graph dataset 1 (listed in Table 5.15) produced by HC, SA, TS drawn within the canvas of our visualization tool

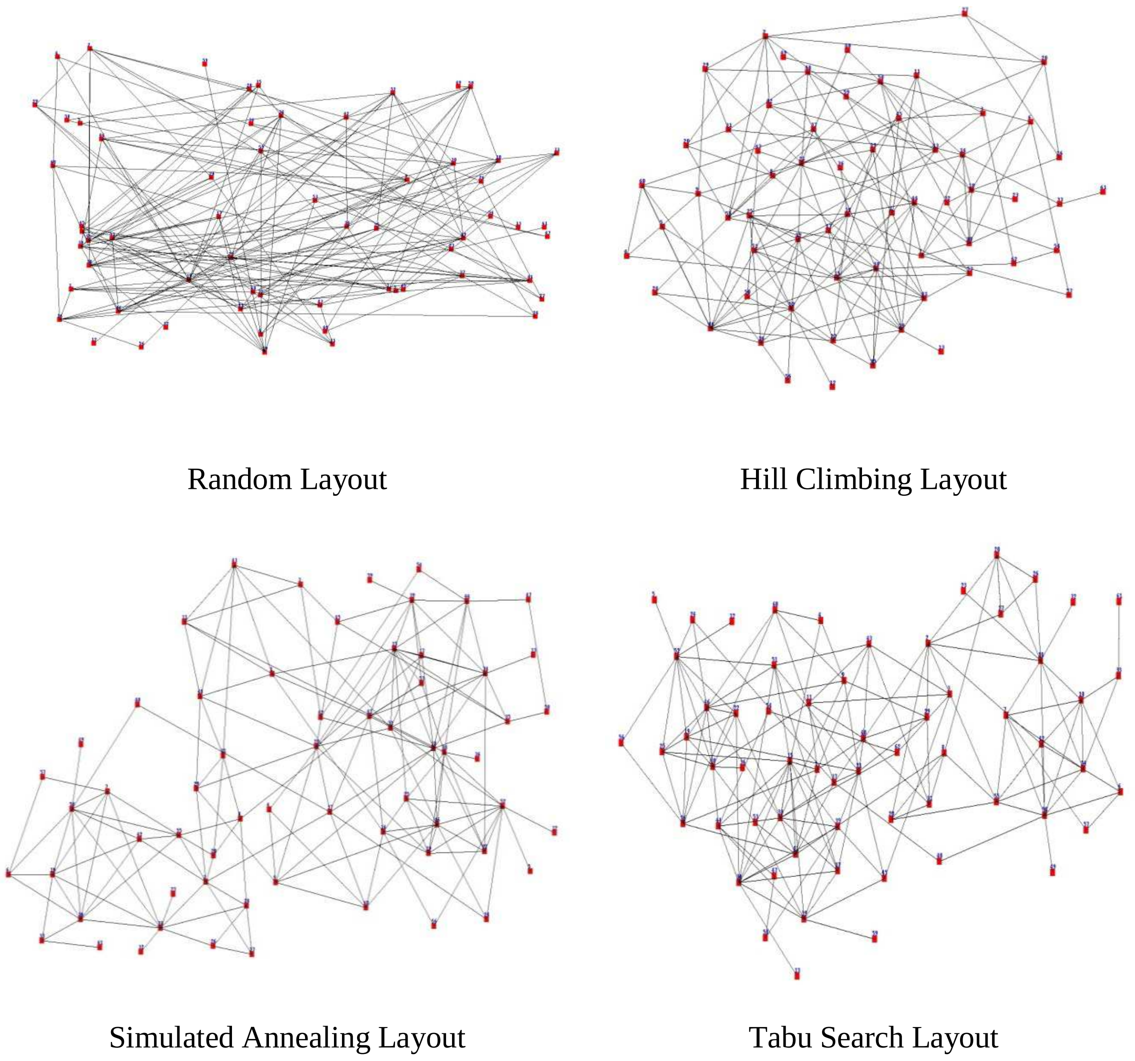


Figure 5.22 Layout of graph dataset 2 (listed in Table 5.15) produced by HC, SA, TS drawn within the canvas of our visualization tool

5.4 Threats to Validity

In terms of threats to validity, two deterministic algorithms and one stochastic algorithm were applied. The deterministic methods ran on the same initial graph layout whereas the stochastic method ran 30 different times on the same initial layout for the same graph. The main internal threat is in the implementation of the algorithms. The three methods were implemented by the same coder, and were run on the same machine. There is the possibility that one of the three methods was implemented in a more efficient way. However, the methods share substantial code that increases confidence that none was particularly disadvantaged.

Another concern is the selection strategy for a neighbour solution to break the tie when two or more neighbour solutions have the same fitness values. In our method, we always break the tie by selecting the solution located on the right. In order to test the significance of the selection strategy, we investigated the number of times in which a tie-break would occur by applying the method on 40 random graphs with a minimum of 60 nodes and a maximum of 110 nodes of different layouts. Table 5.16 shows the average number of occurrences of tie-breaks along with the average total number of solutions. The average percentage of occurrences was below 1% (0.25%) which concludes that the selection strategy is insignificant.

Table 5.16 Average tie-breaks percentage for 40 random graphs

Evaluated Solutions		
Tie-breaks	**Total Solutions**	**Percentage %**
187.5	73758.85	0.25

In terms of external threats, a threat to the generalizability of the results is possible. Selection bias was avoided by using randomly generated graphs (except in the parameters of the generation algorithm, such as number of nodes and edges). However, randomly generated graphs generally do not have the same characteristics as real world graphs hence we also evaluated the methods on real world data sets.

5.5 Summary

In this chapter, we described our research questions and the experiments we performed in order to answer those questions by conducting a comparison between three neighbourhood search-based drawing algorithms: hill climbing, simulated annealing, and tabu search. Our experiments covered the three main aspects of our comparison: how good a layout can be achieved by each drawing algorithm; number of evaluated solutions performed by each method to reach a particular level of layout quality; and quality of layout drawn by the methods after a fixed number of evaluated solutions. The experimental results on random graphs and real world graphs provided quantitative evidence to assert that the tabu search approach can draw a graph with a good layout quality in a lower number of evaluated solutions compared to the hill climbing and the simulated annealing approaches. We also conducted statistical tests which showed, along with the large effect sizes, that the tabu search drawing algorithm was faster than the hill climbing drawing algorithm. It produced (along with simulated annealing) graph layouts with better quality regardless of the graph size in terms of the number of nodes and edges. On the other hand, the efficiency of our tabu search-based method was better than the simulated annealing algorithm but the latter produced graph layouts with similar or slightly better fitness values compared to those produced by our tabu search algorithm when both methods ran without limitations on the number of evaluated solutions. Whilst the tabu search drawing algorithm outperformed the hill climbing drawing algorithm in all aspects and rapidly produced good graph layouts comparable with those produced by the slow simulated annealing, the algorithm has potential to be further improved and so produce better graph layouts if we couple it with methods to more effectively search the problem space, such as path relinking, as we will discuss in Chapter 6.

Chapter 6 Coupling Tabu Search with Path Relinking

This chapter shows the effect of coupling our tabu search-based graph drawing algorithm with path relinking that also belongs to the neighbourhood search-based algorithms. First, we clarify the reason behind specifically choosing path relinking to be coupled with tabu search; second, we highlight our contribution by describing the proposed path relinking-based graph drawing algorithm and showing the process of integrating path relinking within tabu search (Subsection 6.2.1) along with the calibration process of the parameters; third, we discuss different variations of path relinking that can improve the performance of the algorithm, we demonstrate the algorithm of the selected variation (Subsection 6.3.2) along with the tuning process of its parameters; and at last, we give a short summary of the coupling process.

6.1 Why Path Relinking?

The main objective of integrating path relinking within tabu search is to speed up the identification of good solutions. Path relinking is a relatively new neighbourhood search-based method which was originally proposed to improve tabu search and scatter search (Glover et al. 2000). It has proven its efficiency when being coupled with tabu search in many multi-criteria applications as we showed earlier in Chapter 2 (Ho & Gendreau 2006; Peng et al. 2014). However, in a similar manner to tabu search, path relinking has not yet been used in the field of drawing general multi-criteria graph layouts.

In addition to the successful combination of tabu search and path relinking discussed in the literature, there are some other reasons behind selecting tabu search to be coupled with path relinking in particular. Path relinking follows systematic and deterministic rules to combine elite solutions. This is a crucial difference against evolutionary algorithms which use a factor of randomness to create offspring from parent solutions. Stochastic methods could be better than deterministic when they deal with uncertainties. But since we are using fixed values for all the weights in the fitness function in our approach, deterministic methods are favoured as they give the same output when given the same initial layout, unlike stochastic methods when given the same set of parameter values and initial conditions will lead to an ensemble of different outputs (Kleywegt & Shapiro 2001). Consequently, this leads to problems of repeatability which requires running the stochastic method for a number of times for which we calculate the mean or median of the generated outputs. Furthermore, stochastic methods lack good stopping criteria (Kleywegt & Shapiro 2001).

The path relinking procedure takes an initial solution and a guiding solution selected from the set of solutions generated by another search-based method like tabu search. Then the relinking process is applied, where the algorithm aims to gradually introduce the attributes of the guiding solution into the solutions obtained by moving away from the initial solution in a systematic manner. This combination is motivated by the desire to tunnel through blocked off areas and infeasible regions created by the tabu search process (Glover 1997). The tabu list guarantees that the relinking process will only explore solutions which have not been visited in the tabu search process.

6.2 Coupling Tabu Search with Path Relinking for Graph Drawing

Path relinking is a neighbourhood search-based approach which was proposed to intensify and diversify the searching process (Glover & Laguna 1997). It starts with a set of elite solutions that could be generated from other search-based methods such as tabu search, where two solutions are selected from that set: an initial solution and a guiding solution. The relinking process begins from the initial solution and searches in the neighbourhood space for intermediate solutions. These intermediate solutions should introduce more attributes contained in the guiding solution and fewer attributes from the initial solution. The path relinking process usually stops when any of the intermediate solutions reach the guiding solution.

There are different rules discussed in the literature, as shown in Chapter 2 (Section 2.9), for building the set of elite solutions, selecting the initial and guiding solutions, and constructing a systematic and deterministic neighbourhood structure to move along the paths. In the next subsection, we describe how these components were selected and applied in our basic path relinking implementation, in its simplest version, when coupled with our tabu search procedure as an intensification step.

6.2.1 Algorithm

We couple our tabu search procedure with path relinking to intensify the search within a specific space of elite solutions as described in Algorithm 6.1. This algorithm is similar to Algorithm 4.3 plus the steps required for integrating path relinking within tabu search (lines 6, 30-38, and 40 in Algorithm 6.1). The path relinking procedure is called within the tabu search procedure every fixed number of iterations (*intensifyIterations*). Building a reference set of elite solutions is the first step in path relinking. This has a maximum size (*refSize*) and contains no redundant solutions. Unlike the population in genetic algorithms, the reference set in path relinking is recommended to be relatively small (Glover et al. 2000; Ho & Gendreau 2006). Initially, the solutions produced by the tabu search procedure are added to the reference set. A solution is directly added to the reference set as long as the set is not full. However, once the reference set becomes full, a solution will replace the worst solution in the set when any of the following criteria is satisfied:

a. Quality: the fitness value of the added solution is better (smaller) than the fitness value of the best solution in the reference set. This is performed by the *Quality()* function in Algorithm 6.1.

b. Diversity: the fitness value of the added solution is better (smaller) than the fitness value of the worst solution in the set, and it is dissimilar to the solutions in the set. The dissimilarity measure is computed as follows: we define D_s^b, the level of dissimilarity between solution *s* and the best solution *b*, as the sum of distances between the corresponding nodes in the two graph layouts. This is performed by the *Diversity()* function in Algorithm 6.1. We also define the median position of all solutions $x \in$ *refSet* relatively to the best solution *b* as:

$$median\ position = \frac{\sum_{x \in refSet}^{x \neq b} D_x^b}{|refSet| - 1}$$

where |*refSet*| denotes the number of solutions in the reference set. A solution *s* is included in *refSet* if its fitness value is better than the fitness value of the worst solution in *refSet* and its level of dissimilarity exceeds the median, $D_s^b > Median$.

Given:

Connected Graph G(V,E): V is a set of nodes and E ⊆ (V×V) is a set of edges.

`initialSquareSize`: predefined square size where tabu search candidate solutions are located on its border.

`squareReduction`: predefined value which represents the rate of reduction for the size of the square.

`maxIterations`: predefined maximum number of iterations of the tabu search drawing algorithm.

`initialCutOff`: predefined minimum value that determines whether a move is tabu or not.

`intensifyCutOff`: predefined value which represents the rate of reduction for the current `cutOff` value.

`intensifyIterations`: predefined number of iterations in which the tabu search searching process starts to intensify.

`duration`: predefined number of iterations in which a move should remain in the tabu list.

`refSize`: predefined size for the maximum number of solutions that can be added to the reference set of path relinking.

Algorithm:

```
allOffsets = {(1,1), (1,0), (1,-1), (0,-1), (-1,-1), (-1, 0), (-1, 1), (0, 1)}
tabuSet = {}
squareSize = initialSquareSize , CutOff = initialCutOff
layout = RandomizeLayout(G) /* layout maps each node in G to an (x,y) position */
iteration = 0
refSet = {}    /* PR empty reference set */
while iteration < maxIterations do
 for v in V do
  currentPos = layout[v] /* position currently associated with node v */
  currentFitness = Fitness(layout)
  candidates = {}
  for scaledOffset in {(squareSize*x, squareSize*y) | (x,y) in allOffsets}
   candidatePos = currentPos + scaledOffset /* vector addition */
   if (v, candidatePos, i) ∉ tabuSet for some i then
    layout[v] = candidatePos
    candidateFitness = Fitness(layout)
    if candidateFitness / currentFitness > CutOff then
     tabuSet = tabuSet ∪ {(v, candidatePos, iteration)}
    else
     candidates = candidates ∪ {(candidatePos, candidateFitness)}
    end if
   end if
  end for
  if candidates ≠ {} then
   newPos = p, where (p,f) is the pair in candidates with minimal f
   layout[v] = newPos
   tabuSet = tabuSet ∪ {(v, currentPos, iteration)}
  end if
 end for
```

```
  if !FoundinRefSet(layout) then
    if Size(refSet) < refSize then /* not full */
      refSet = refSet ∪ {(layout, iteration)}
    else
      if Quality(layout) || Diversity(layout) then
        refSet = refSet ∪ {(layout, iteration)}
      end if
    end if
  end if
  if (iteration mod intensifyIterations) == 0 then
    layout = PathRelinking(refSet, iteration)
    squareSize = SmallerSquareSize(squareSize, squareReduction)
    cutOff = SmallerTabuCutOff(cutOff, intensifyCutOff)
  end if
  tabuSet = {(v,p,i) | (v,p,i) in tabuSet and (iteration - i) < duration}
  iteration = iteration + 1
end while
```

Algorithm 6.1 Tabu search and path relinking coupling algorithm for graph drawing

When the path relinking procedure is called, the following steps are performed for a set number of iterations (*PRmaxIterations*) as long as the reference set has more than one solution (see Algorithm 6.2): firstly, we select two solutions from the reference set (initial and guiding solutions). There are different ways for selecting these two solutions as we show later in this chapter. In our first version of this algorithm, we select the worst and the best solutions from the reference set to represent the initial and guiding solutions respectively, i.e. the guiding solution is always of a better (smaller) fitness value than the fitness value of the initial solution. Secondly, we remove the initial solution from the reference set as its path to the guiding solution will be explored. Thirdly, we call the function *MoveAlongPath()* that moves on a path from the initial solution toward the guiding solution and vice versa in the solution space to generate intermediate solutions (see Algorithm 6.3). This scenario had produced better results in other applications compared to moving in one direction only (Ho & Gendreau 2006). These intermediate solutions should become closer to the guiding solution (i.e. contain more attributes from the guiding solution and fewer attributes from the initial solution). In our algorithm, for each node in the initial solution, we visit the 8 positions around a square (same local search space described earlier in Chapter 4, Section 4.2.1) of a predefined size (*pathSqrSize*) and compute the Euclidean distance from each position to its corresponding node in the guiding solution, as shown in Figure 6.1 where node number 2 would move to a neighbourhood node that has the closest Euclidean distance to its corresponding node in the guiding solution. We select the position with the shortest Euclidean distance. Its fitness value is computed along with its dissimilarity level, and we update the reference set, by calling function *UpdateReferenceSet()*, if the new solution satisfies the quality and dissimilarity measures. The movement along the path requires two conditions to stop: the first is when an intermediate solution reaches the guiding solution, and the second is when the length of the path reaches a predefined value of a maximum length (*pathLength*). Note that, as we generate intermediate solutions, we use the tabu search memory-based list to avoid previously visited solutions.

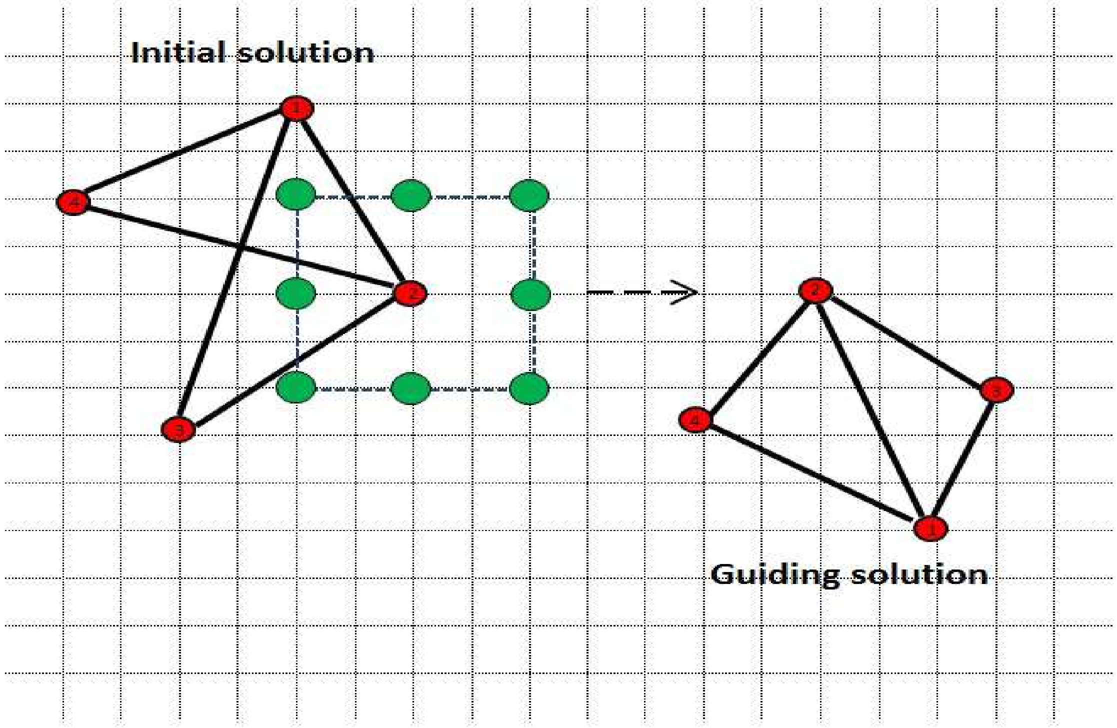


Figure 6.1 Our path relinking strategy in moving from the initial solution to the guiding solution

Given:

`PRmaxIterations`: predefined value of the number of iterations to repeat the path relinking procedure.

`pathSqrSize`: predefined square size where path relinking candidate solutions are located on its border.

`pathLength`: predefined value representing the maximum length of the path.

Algorithm:

```
1: i = 0
2: while i < PRmaxIterations && Size(refSet) > 1 do
3:  SelectSourceDestination(refSet, source, destination)
    /* returns source and destination selected from the reference set refSet */
4:  candidateLayout1 = MoveAlongPath(source, destination, pathLength, pathSqrSize)
    /* forward path */
5:  candidateLayout2 = MoveAlongPath(destination, source, pathLength, pathSqrSize)
    /* backward path */
6:  UpdateReferenceSet(refSet, Min(candidateLayout1, candidateLayout2))
7:  i = i + 1
8: end while
```

Algorithm 6.2 PathRelinking() procedure

Algorithm:

```
MoveAlongPath (source, destination, pathLength, pathSqrSize)
1: length = 0
2: while source != destination || length < pathLength
3:  for v in V do
4:    position = ShortestEuclidean(source[v], destination[v], pathSqrSize)
    /* position with shortest distance around the square from the node in source to
       destination */
5:    fitness = Fitness(layout[position])
6:    move source[v] to position if position ∉ tabuSet
7:    tabuSet = tabuSet ∪ {(v, layout[position], iteration)}
8:  end for
9:  length = length + 1
10:end while
```

Algorithm 6.3 MoveAlongPath() procedure

6.2.2 Parameter Tuning

Our simplest version of path relinking has four parameters which affect the performance of the method: the number of times we pick initial and target solutions from the reference set for path testing (*PRmaxIterations*), the size of the reference set (*refSize*), the maximum length of the path between the initial and the target solutions (*pathLength*), and the size of the square used to determine the neighbourhood search space of solutions in the path (*pathSqrSize*). In this subsection, we try to calibrate the values of the parameters of the path relinking procedure while fixing the values of the tabu search procedure to the ones we obtained earlier in Chapter 4 (Section 4.5.2). Note that, we could have re-calibrated the values of parameters for tabu search, but we moved on since tabu search does not have an effect on the parameters of path relinking (Ho & Gendreau 2006) as path relinking is a separate function for search intensification. Tabu search is only responsible for building the reference set used in path relinking by adding elite solutions to the set.

The graph datasets which we used in tuning the values of these parameters were exactly the same sets used in tuning the values of the parameters of all the previous methods as described in Table 6.1, i.e. 100 random connected graphs, based on Erdos-Renyi model, that were divided into five sets such that each set had a different number of nodes and edges.

Table 6.1 Graph datasets used in parameter tuning for path relinking

Graph Set	Nodes	Edges	Density	Label
1	50	153	0.125	N50E153
2	100	544	0.110	N100E544
3	150	1173	0.105	N150E1173
4	200	1890	0.095	N200E1890
5	250	2645	0.085	N250E2645

We followed the same incremental testing process we performed with all the other methods. The process was divided into three phases. In phase I, we select arbitrary values for all parameters then we test each parameter with several values. We start with one parameter, we test it thoroughly with different values, and then we select the value that produces the best layout compared to the other values. We fix the value of the first parameter and we move on to test another parameter in the same manner, and so forth. In the second phase of parameter tuning, we repeated the same steps we followed in phase I, but instead of starting with arbitrary values, we started with the values that were selected and fixed from phase I. The third phase is to study the effect of the values of the path relinking parameters on the performance of the drawing algorithm (i.e. number of evaluated solutions).

i. Phase I

At the beginning, we tested different values for the *PRmaxIterations* parameter {1, 2, 3, 4, 5}, while fixing the rest of the parameters to some arbitrary values*: refSize = 5*, *pathLength = 10*, and *pathSqrSize = 10*. In this phase of testing, we were looking for the combination of parameters' values that give the smallest fitness value (best quality) compared to all other combinations regardless of the number of evaluated solutions performed by the drawing algorithm. According to Figure 6.2, we selected the value 5 for the *PRmaxIterations* parameter. Note that we could increase the values for this parameter as the figure shows that the fitness value decreases as the number of iterations increases. However, we stopped at the value 5 since the arbitrary value of the maximum size of the reference set is small. After testing the *refSize* parameter we can choose larger values for *PRmaxIterations* in phase II.

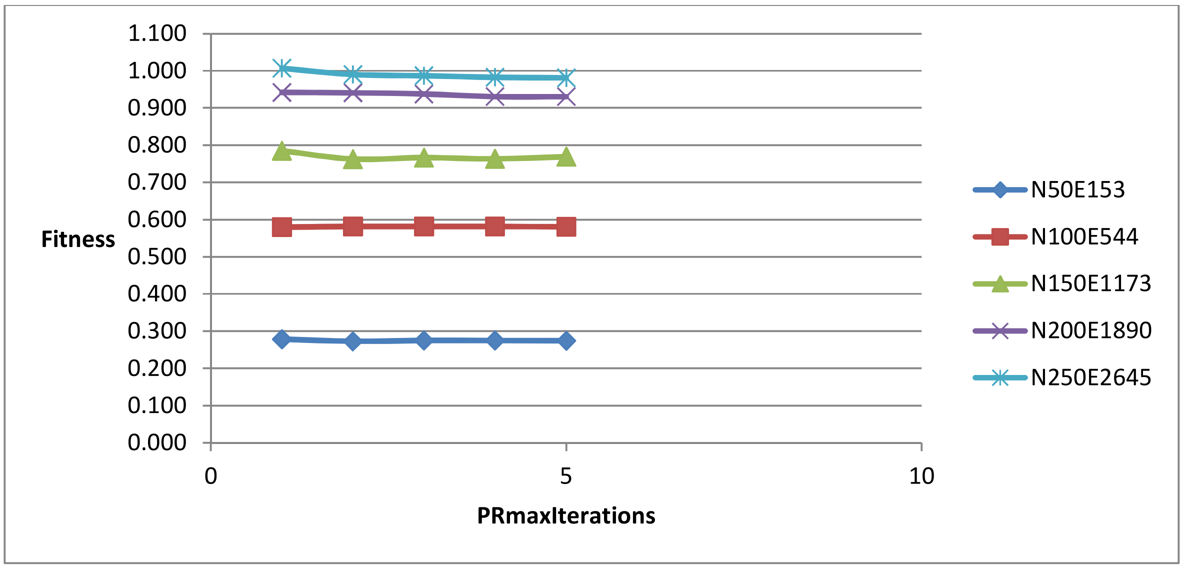


Figure 6.2 Path relinking fitness with the PRmaxIterations parameter (phase I)

After fixing the value of *PRmaxIterations*, we moved on to the second parameter *refSize* and tested it with the values {5, 10, 15, 20, 25}. As shown in Figure 6.3, the layout became better as the size of the reference set increased. When *refSize* was assigned the value 20 or 25, it produced better graph layouts compared to the rest of the values. But we selected the value 20 for this parameter as the number of evaluated solutions was lower.

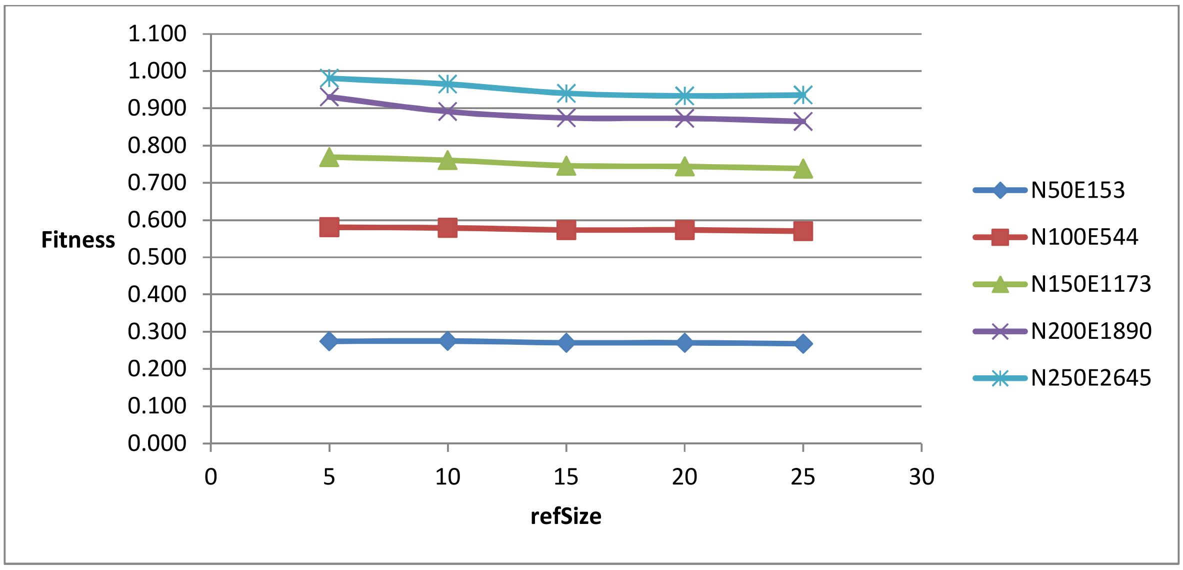


Figure 6.3 Path relinking fitness with the refSize parameter (phase I)

The maximum length of the path between the initial solution and the guiding solution, *pathLength* was tested with the values {5, 10, 15, 20, 25}. The fitness values of the layouts produced by our path relinking drawing algorithm were close to each other, as shown in Figure 6.4. However, on large graph datasets whether we decrease or increase the value of this parameter around the value 15, the fitness value increases. Thus, we selected the value 15 for *pathLength* as the figure shows that this value gave the best layout.

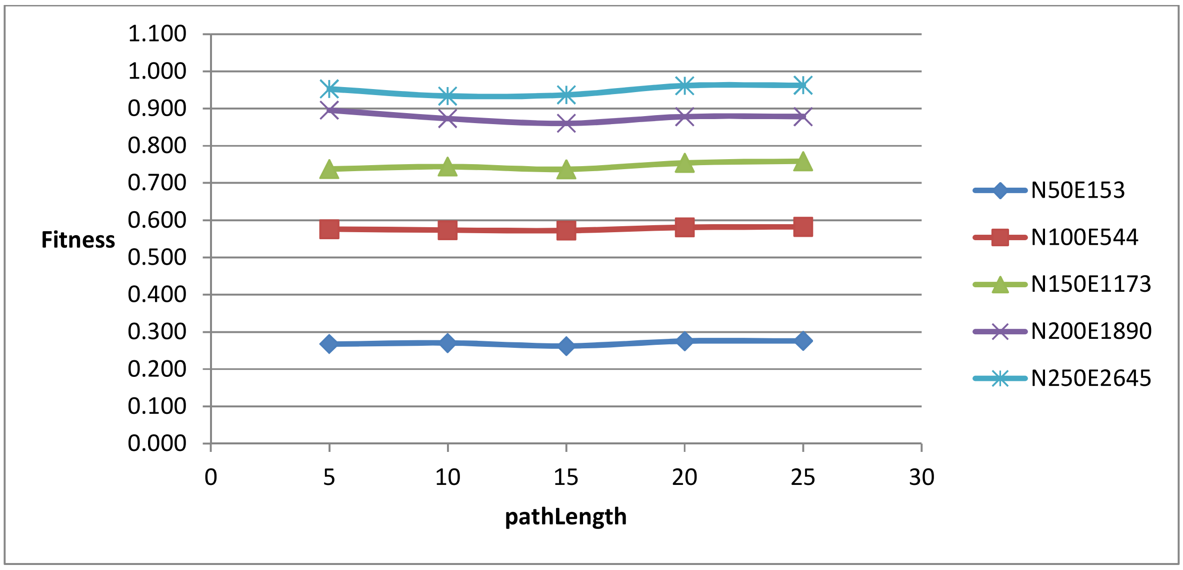


Figure 6.4 Path relinking fitness with the pathLength parameter (phase I)

The square size representing the neighbourhood search space around each solution in the path was tested with the values {2, 6, 10, 14, 18}. Figure 6.5 shows that the fitness value decreased as the value of *PathSqrsize* increased until the value 14 was reached. After that the fitness value had increased again at the value 18. We chose the value 14 for this parameter since it produced the best fitness values of graph layouts when applied on large graphs.

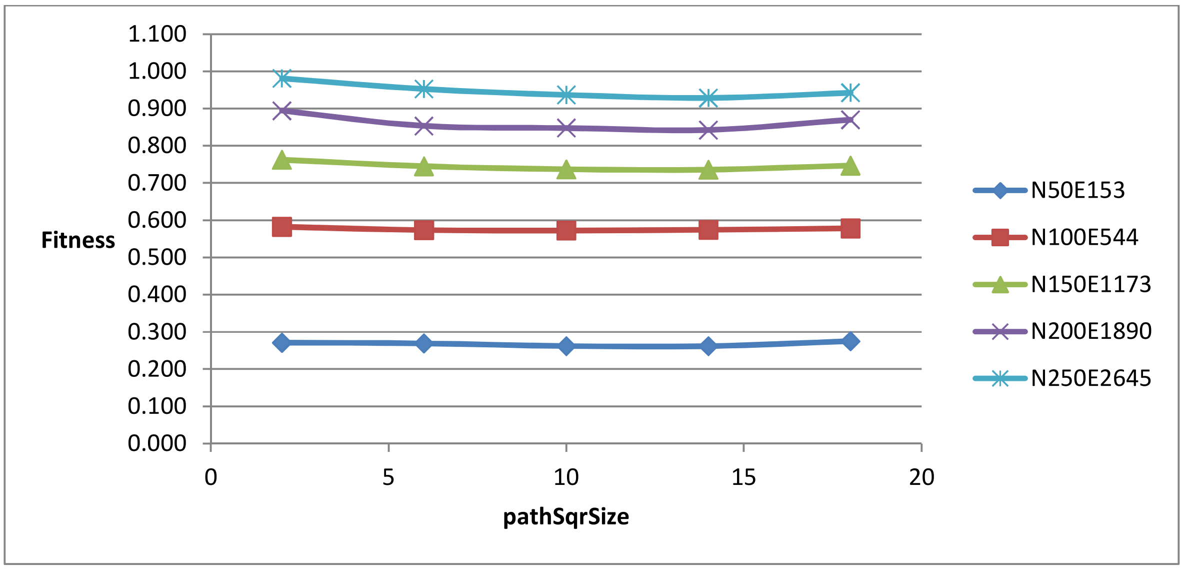


Figure 6.5 Path relinking fitness with the pathSqrSize parameter (phase I)

ii. Phase II

We started with the values that were selected and fixed from phase I. In Figure 6.6, we show the results of testing the value of *PRmaxIterations* with the values {3, 5, 7, 9, 11}. The figure shows that the best layouts were produced when the value of this parameter was either 9 or 11. But we selected the value 11 as it produced a slightly better fitness value compared to the layout's fitness value produced when the value 9 was used on large graphs.

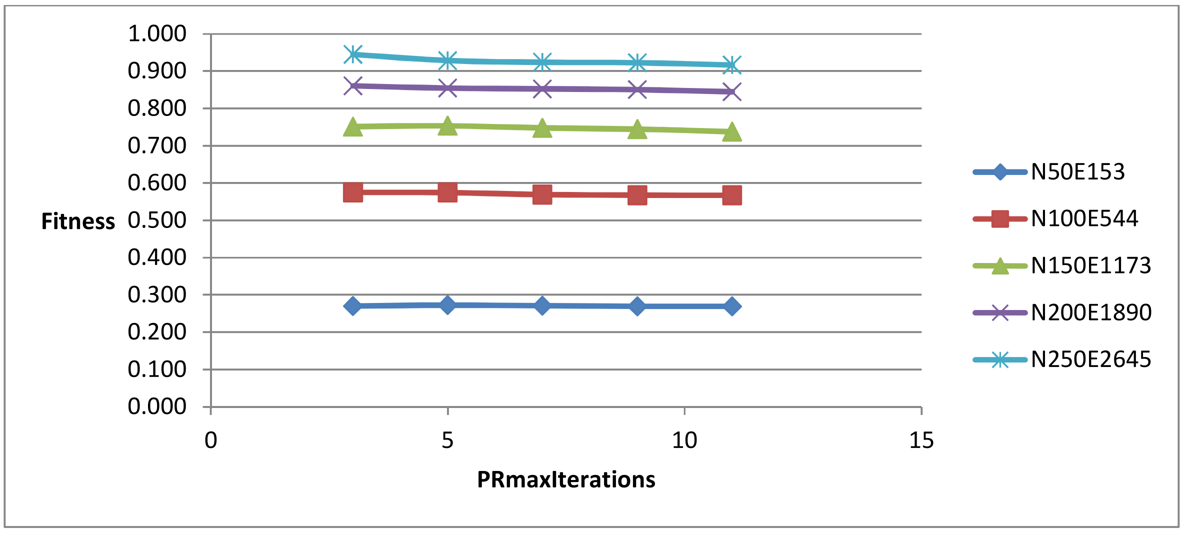


Figure 6.6 Path relinking fitness with the PRmaxIterations parameter (phase II)

In this phase, there is no significant difference between the fitness values produced when we tested *refSize* parameter with the values {10, 15, 20, 25, 30} except with large graphs as Figure 6.7 indicates that the value 25 for this parameter produced layouts with the best fitness value compared to the other values of the parameter.

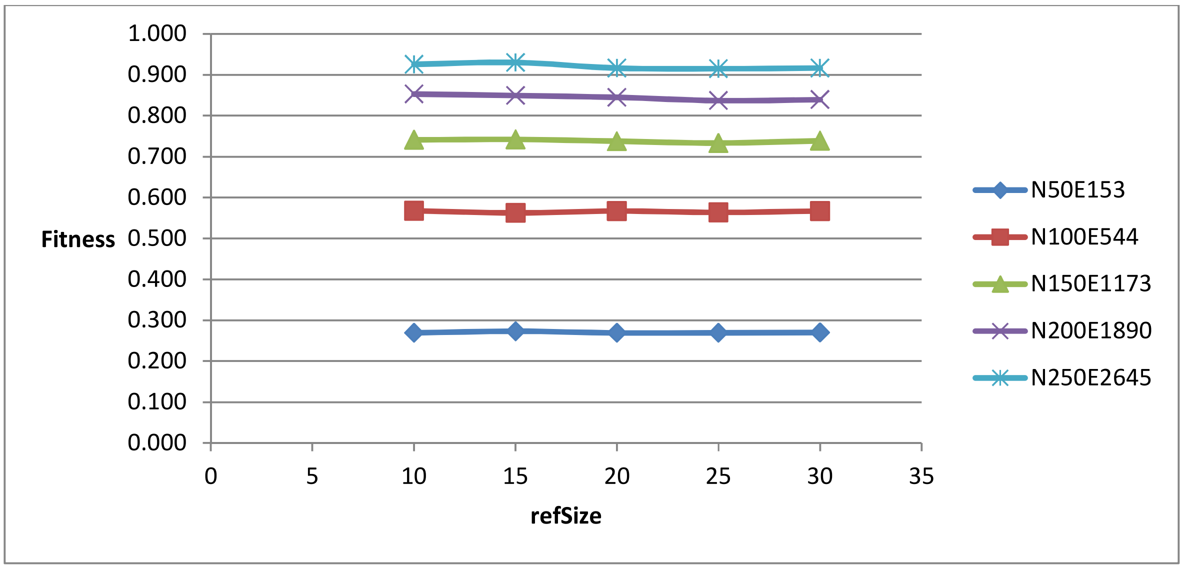


Figure 6.7 Path relinking fitness with the refSize parameter (phase II)

We retested the value of the *pathLength* parameter in this phase using the values {10, 15, 20, 25, 30}. Figure 6.8 shows once more that the best fitness values were produced at value 15 for all the categories of graphs used in this experiment. The figure also shows that the fitness values increased as the length of the path increased starting from the value 15.

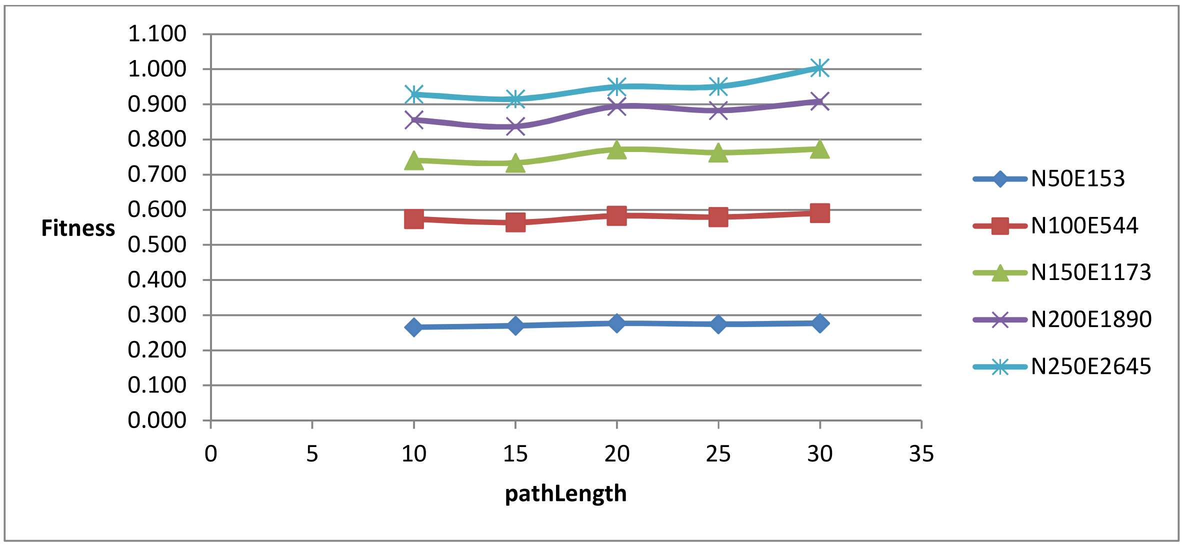


Figure 6.8 Path relinking fitness with the pathLength parameter (phase II)

The results in phase I showed that increasing the value of the *pathSqrSize* parameter decreases the value of the layout's fitness until the value of this parameter reaches the value 14. In this phase, we retested this parameter with the values {4, 9, 14, 19, 24}. Figure 6.9 shows that phase II gave similar results to those generated in phase I. The fitness values were at their best when the value of this parameter was 14.

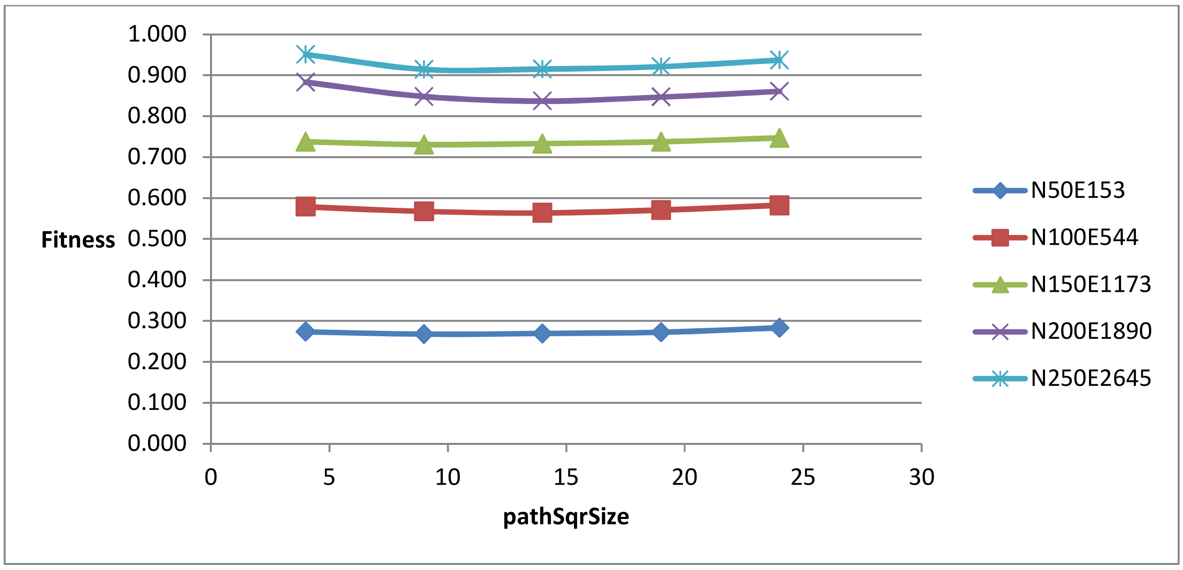


Figure 6.9 Path relinking fitness with the pathSqrSize parameter (phase II)

iii. Phase III

We performed a similar process to the one used in parameter tuning of the previous drawing algorithms discussed in Chapter 4. We took a view that a good-enough graph layout is a layout in which its fitness value is slightly greater than the best fitness value produced in the experiments of phase II. We used the values of the fitness function produced by the selected parameters' values in phase II and we increased them by 12.5%. Then we ran the path relinking procedure until it reached equal fitness values to the target fitness values or no further improvement in the fitness value was made. Finally, we selected the most appropriate parameter values that gave a good layout with a small number of evaluated solutions.

The best value for the *PRmaxIterations* parameter in phase II was 11. As we are looking to minimise the number of evaluated solutions in phase III, we tested this parameter with values smaller than 11: {4, 5, 7, 9}. The results in Table 6.2 and Figure 6.10 indicate that the method produced layouts with fitness values equal or smaller than the targeted fitness values with all the values of *PRmaxIterations* which we tested. However, the value 4 was the one which made the method generate the targeted layouts with the lowest number of evaluated solutions. To ensure that there is no smaller value for *PRmaxIterations* which could produce results better than the value 4, we tested it with the value 2 instead. But this value could not allow the method to reach the targeted fitness value for one of the tested graph datasets, as shown in Table 6.2.

Table 6.2 Path relinking fitness with the PRmaxIterations parameter (phase III)

	Fitness				
PRmaxIterations	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
2	0.300	0.626	0.813	0.944	1.017
4	0.299	0.624	0.809	0.926	1.009
5	0.296	0.622	0.808	0.926	1.008
7	0.298	0.621	0.811	0.924	1.007
9	0.299	0.619	0.810	0.926	1.006
Target	0.303	0.634	0.825	0.941	1.029

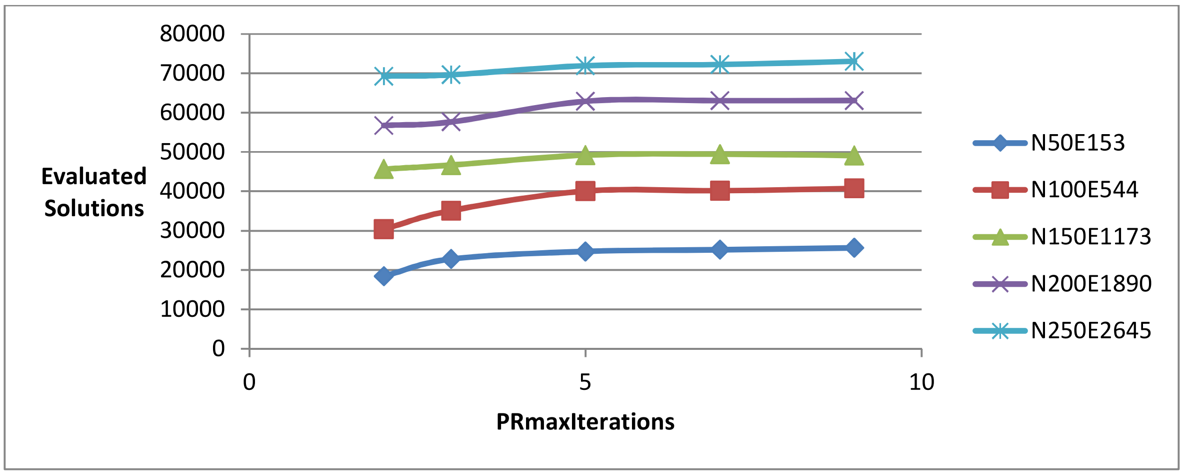


Figure 6.10 Path relinking number of evaluated solutions with the PRmaxIterations parameter (phase III)

In the previous phase, the best value for the reference set size parameter was 25. We tested the *refSize* parameter in this phase with the values {5, 10, 15, 20, 25}. Figure 6.11 shows that there was no significant difference in the number of evaluated solutions when we tested this parameter with the values ranging from 10 to 25. Only the value 5 had generated a lower number of evaluated solutions. However, a path relinking procedure with a reference set of size 5 could not generate graph layouts with fitness values that reach the targeted fitness values in two graph datasets as shown in Table 6.3. We decided to select the value 20 for the *refSize* parameter as the fitness values produced by the method were better and smaller than the targeted fitness values with a similar number of evaluated solutions compared to the other values of this parameter.

Table 6.3 Path relinking fitness with the refSize parameter (phase III)

	Fitness				
refSize	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
5	0.305	0.627	0.814	0.933	1.030
10	0.302	0.623	0.814	0.936	1.024
15	0.301	0.624	0.814	0.930	1.020
20	0.300	0.625	0.805	0.928	1.009
25	0.300	0.625	0.807	0.929	1.011
Target	0.303	0.634	0.825	0.941	1.029

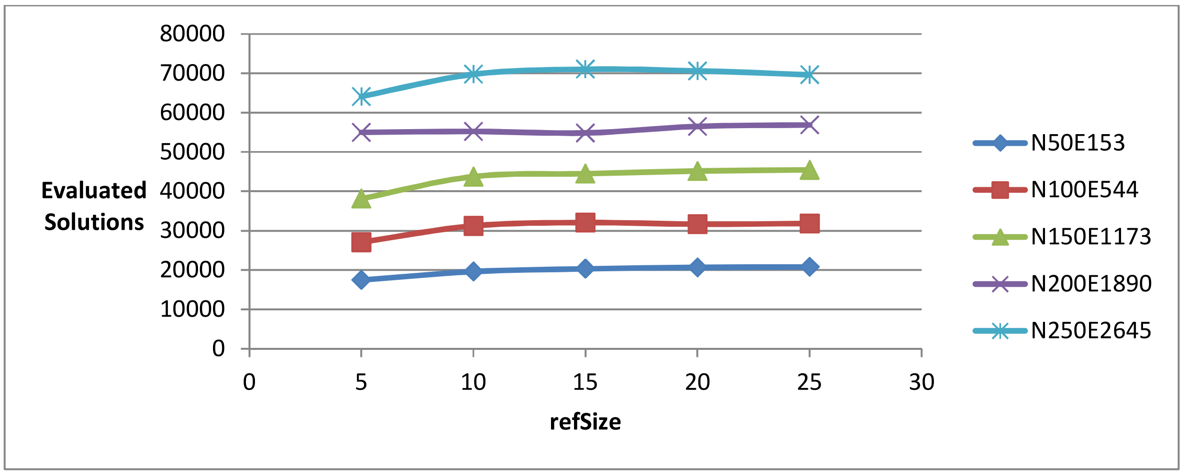


Figure 6.11 Path relinking number of evaluated solutions with the refSize parameter (phase III)

According to the tuning process for the *pathLength* parameter in phase II, we obtained the best fitness values when the value of this parameter was 15. As this value increased or decreased, the fitness values were also becoming larger (worse). The length of the path was tested in phase III with the values {5, 10, 15, 20, 25}. Figure 6.12 illustrates that the number of evaluated solutions decreases as the value of *pathLength* decreases. A path length with the value 5 produced a lower number of evaluated solutions compared to the other values, but the algorithm could not reach the targeted fitness value for one graph dataset, as shown in Table 6.4. Thus, the value 10 was the best value for *pathLength* in which it produced graph layouts having similar fitness values to the targeted fitness values with a lower number of evaluated solutions compared to the other values of this parameter.

Table 6.4 Path relinking fitness with the pathLength parameter (phase III)

	Fitness				
pathLength	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
5	0.299	0.636	0.812	0.929	1.018
10	0.299	0.626	0.807	0.929	1.014
15	0.300	0.625	0.805	0.928	1.009
20	0.300	0.627	0.819	0.947	1.023
25	0.299	0.626	0.816	0.943	1.031

Target	0.303	0.634	0.825	0.941	1.029

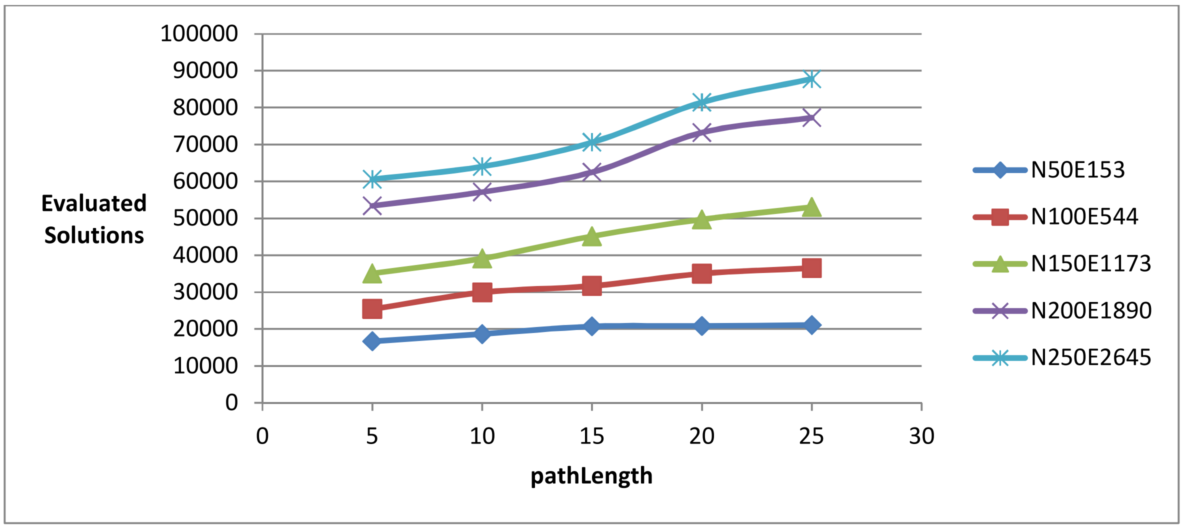


Figure 6.12 Path relinking number of evaluated solutions with the pathLength parameter (phase III)

In phase II, the best value for *pathSqrSize* was 14. In this phase, we retested this parameter with the values {6, 10, 14, 18, 22}. Table 6.5 shows that the path relinking procedure could reach the targeted fitness values with all the values we tested for *pathSqrSize*. On the other hand, Figure 6.13 shows that the number of evaluated solutions decreased as the square size increased. The figure shows that starting from the value 18 onwards, the number of evaluated solutions becomes stable. Therefore, we chose the value 18 for the *pathSqrSize* parameter.

Table 6.5 Path relinking fitness with the pathSqrSize parameter (phase III)

	Fitness				
pathSqrSize	**N50E153**	**N100E544**	**N150E1173**	**N200E1890**	**N250E2645**
6	0.302	0.624	0.813	0.927	1.014
10	0.294	0.625	0.810	0.932	1.016
14	0.299	0.626	0.807	0.929	1.014
18	0.300	0.622	0.812	0.928	1.019
22	0.300	0.628	0.816	0.930	1.024
Target	0.303	0.634	0.825	0.941	1.029

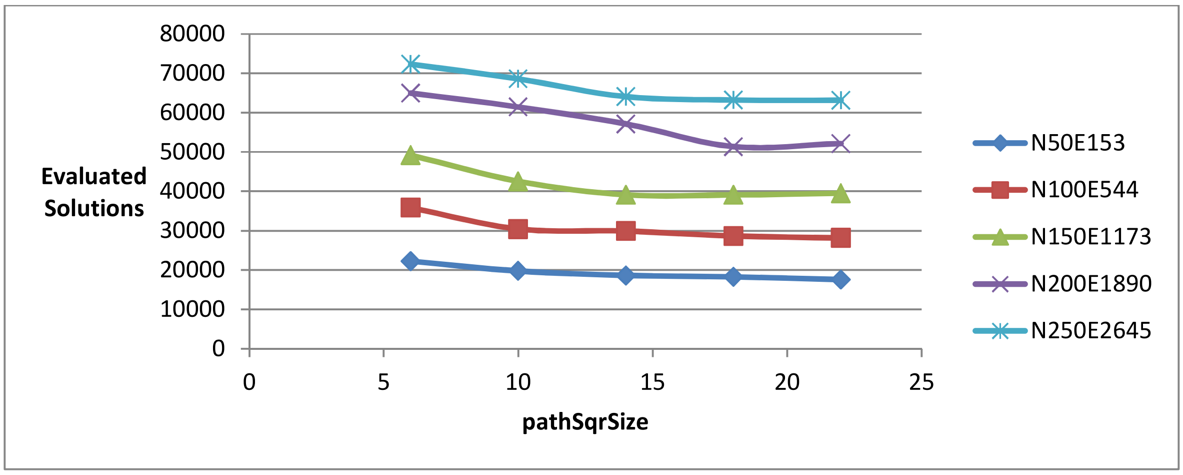


Figure 6.13 Path relinking number of evaluated solutions with the pathSqrSize parameter (phase III)

Now that we have tuned the parameters of the simplest version of path relinking for graph drawing, we list the values for the parameters of the path relinking procedure below that we will use in the coming experiment:

PRmaxIterations = 4,

refSize = 20,

pathLength = 10,

pathSqrSize = 18.

In the following section, we discuss different variations of path relinking that could improve the performance of the path relinking procedure in its simplest version.

6.3 Variation of Path Relinking

The performance of the path relinking procedure is influenced by the strategy used for selecting the initial and the guiding solutions. It is also affected by the technique used in searching for solutions in the neighbourhood search space. In this section, we describe the different strategies applied in order to select the best variation of path relinking that improves the basic implementation in terms of the performance and quality of the produced layouts. We also include experimental results which second our selections.

6.3.1 Proper Selection of Initial and Guiding Solutions

Different selection strategies for the initial (source) and guiding (destination) solutions affect the quality of the graph layouts drawn by the path relinking procedure. There are five different variations for the selection mechanism of the source solution and the destination solution from the reference set (Ho & Gendreau 2006):

a. The worst and the best elite solutions,

b. The best and the second best elite solutions,

c. Random selection of elite solutions,

d. The best elite solution and the most distant elite solution to the best. In our graph layout application, the distance between two layouts can be computed as the summation of Euclidean distances between the corresponding nodes in the two layouts as described in the `Diversity()` function used in Algorithm 6.1 which was discussed in Section 6.2.1. The most distant solution is the one with the maximum summation of distances to the best elite solution (i.e. the most distant solution = *s* such that $s \in refSet$ and satisfies the formula: $max \sum_{s \in refSet}^{s \neq b} D_s^b$, where *b* is the best solution in *refSet* and D_s^b is the level of dissimilarity between solutions *s* and *b*),

e. The two most distant elite solutions.

In our basic version of the path relinking procedure, we started with the first strategy where source and destination solutions were the worst and the best elite solutions in the reference set. But as we want to choose the variation that gives the best performance, we tested the five different strategies on random connected graph datasets, as shown in Table 6.6. We generated 40 random graphs, based on Erdos-Renyi model, divided into 4 groups such that each group contains 10 test cases. Each group had a number of nodes and a number of edges that varies from the number of nodes and edges in the other groups. The results in Figure 6.14 and Figure 6.15 show that the first (a) and the fourth (d) strategies were competitive and had better performance compared to the other strategies, taking into consideration the combination of both quality and speed.

Table 6.6 Characteristics of the graphs used in the experiment of selecting initial/guiding solutions

Graph Set	Nodes	Edges	Density
1	55	190	0.128
2	105	611	0.112
3	155	1217	0.102
4	205	1986	0.095

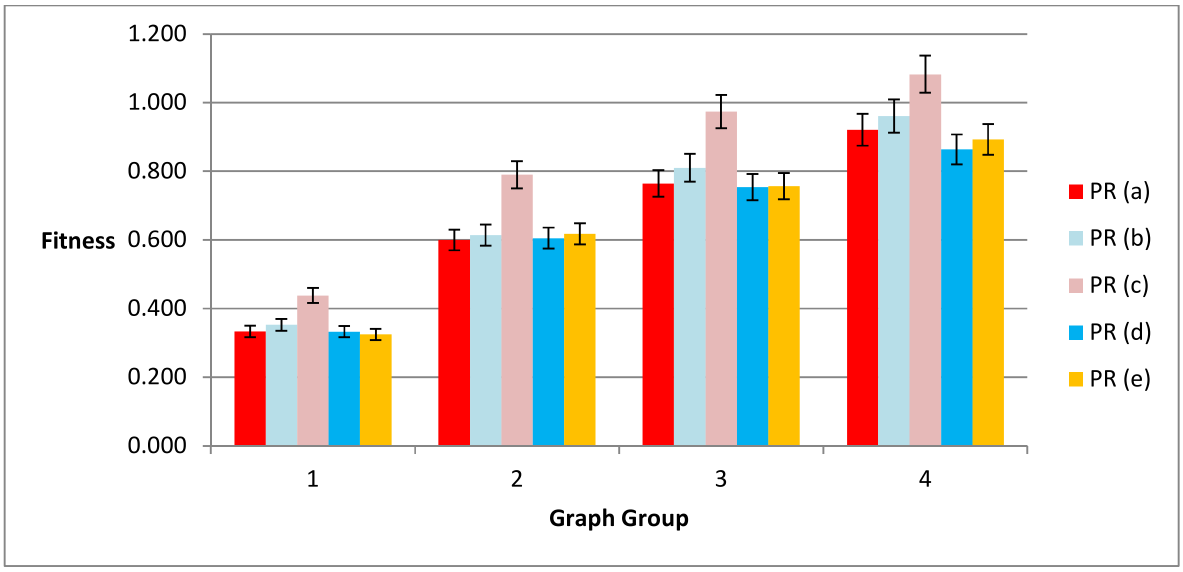


Figure 6.14 Fitness values with 95% confidence interval of the strategies for selecting initial/guiding solutions

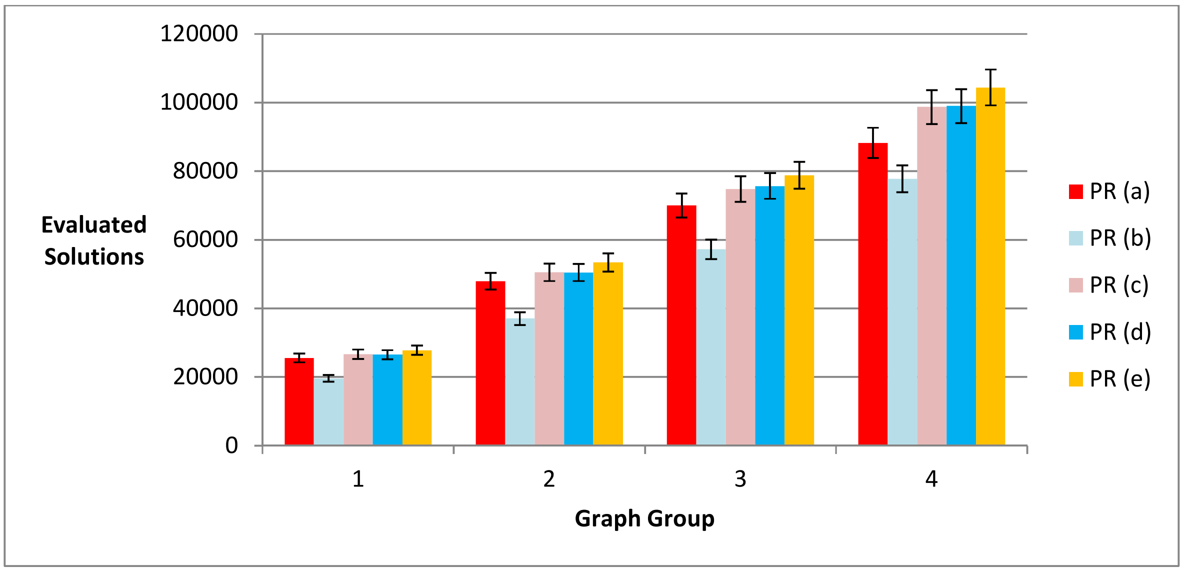


Figure 6.15 Number of evaluated solutions with 95% confidence interval of the strategies for selecting initial/guiding solutions

We performed another comparison between those two strategies on newly generated random datasets (described in Table 6.7) in order to avoid overfitting. The graph datasets were divided into groups in the same way which we followed in the previous experiment. We divided the comparison into two phases: the first phase tested the number of evaluated solutions performed by the drawing algorithm for each strategy as it runs until it reaches a set fitness value; and the second phase tested the quality of the generated graph layouts when the drawing procedure runs for a set number of evaluated solutions. In both phases, the path relinking procedure which was based on strategy (d) slightly outperformed the procedure with strategy (a) in terms of speed as shown in Figure 6.16 and the fitness of the generated layouts as shown in Figure 6.17.

Table 6.7 Characteristics of the graphs used in the experiment of comparing strategies (a) and (d) for selecting solutions

Graph Set	Nodes	Edges	Density
1	150	1173	0.105
2	200	1890	0.095
3	250	2645	0.085
4	300	3363	0.075

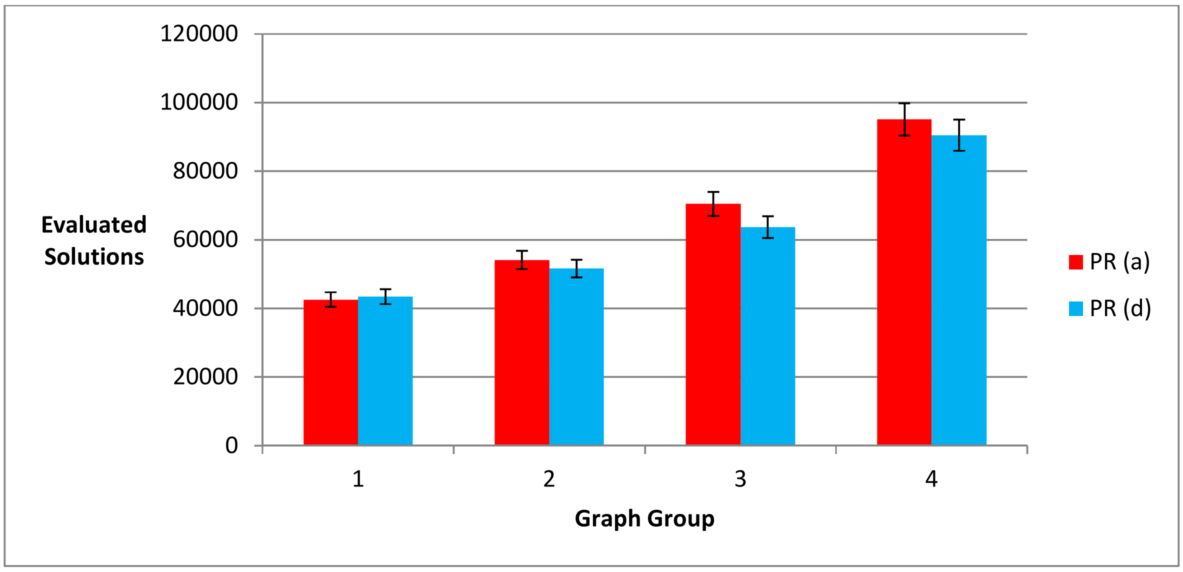


Figure 6.16 Number of evaluated solutions with 95% confidence interval performed when strategies (a) and (d) run to reach a set fitness value

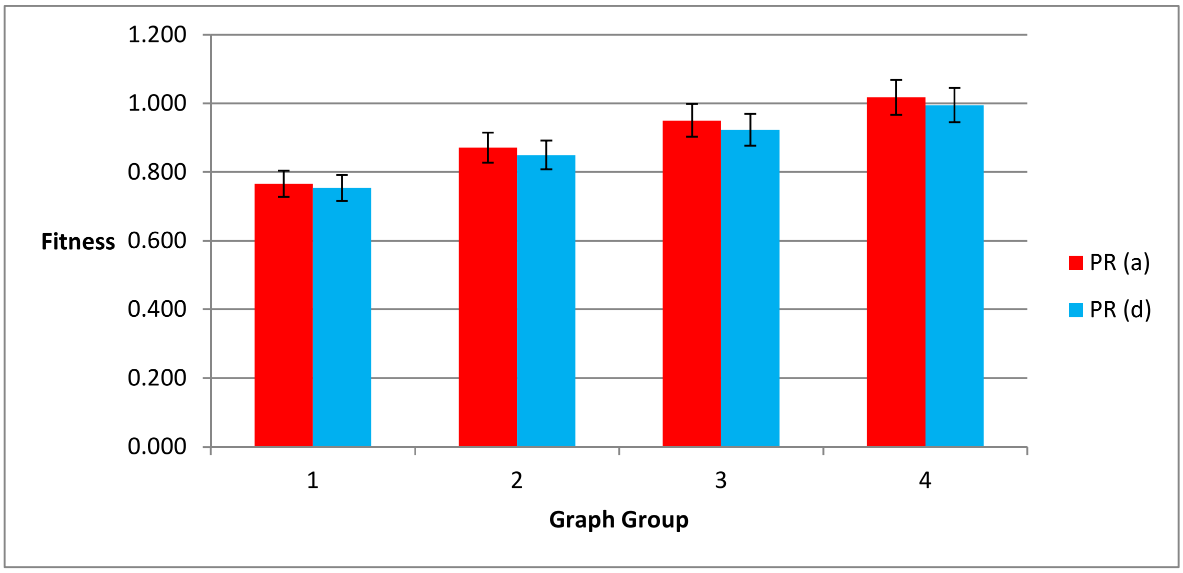


Figure 6.17 Fitness values produced with 95% confidence interval when strategies (a) and (d) run for a set number of solutions

6.3.2 Improved Neighbourhood Searching Strategy

After improving our basic implementation of the path relinking procedure by choosing the strategy, for selecting the initial and the guiding solutions, that works best with our graph drawing application, we proposed another improvement to our path relinking procedure by examining the way the path is formed from the initial solution to the guiding solution.

In the basic implementation, the step-size we use to move from an initial solution to intermediate solutions is fixed (*pathSqrSize*) and it never changes as we move along the path until we reach the guiding solution. We examined if using a variable step-size would improve performance. Moving along the path such that the movement starts faster near the initial solution and it becomes slower as it gets closer to the guiding solution in the solution space which intensifies the search in the area of the guiding solution. This strategy is applied to both directions: from an initial solution to a guiding solution and vice versa. This variation introduces two new parameters to our path relinking procedure: number of iterations required to update the step-size (*accelerationPeriod*), and the rate of decreasing the step-size (*accelerationRate*). The net effect is to search more closely to the two known solutions than in the space between them. Note that, moving in a variable step-size will not exclude the solutions in the middle of the path. They will have a fair exploration time, as the acceleration takes place at one end and slows down at the other end in both directions. However, according to Sánchez-Oro & Duarte (2012), the best solutions were usually detected near the guiding solution since the main purpose of path relinking is intensifying the search near elite solutions.

Before we compare between those two strategies (fixed step-size or variable step-size), we need to select proper values for the newly introduced parameters while fixing the other parameters of path relinking to the values which were determined earlier in Section 6.2. Therefore, we firstly chose initial arbitrary values for those parameters and we performed a tuning process on those values by applying the method on randomly generated graph layouts. We generated 50 random graphs, based on Erdos-Renyi model, split into 5 groups, as shown in Table 6.8, such that each group contains 10 test cases.

Table 6.8 Characteristics of the graph datasets used for choosing proper values for the acceleratioPeriod and accelerationRate parameters

Graph Set	Nodes	Edges	Density	Label
1	50	156	0.128	N50E156
2	100	549	0.111	N100E549
3	150	1173	0.105	N150E1173
4	200	1970	0.099	N200E1970
5	250	2583	0.083	N250E2583

We performed two rounds of tuning the values of *accelerationPeriod* and *accelerationRate*. In the first round, we fixed the value of *accelerationRate* to 0.01, and we examined a set of values for the other parameter {1, 5, 10, 15, 20}. We were looking for the value that gives the best fitness compared to the other values and if we get a tie, we select the one that gives a lower number of evaluated solutions. The line charts in Figure 6.18 and Figure 6.19 show that the fitness value and number of evaluated solutions performed by the drawing algorithm become stable after the value 10 that we selected as a value for *accelerationPeriod* in this round.

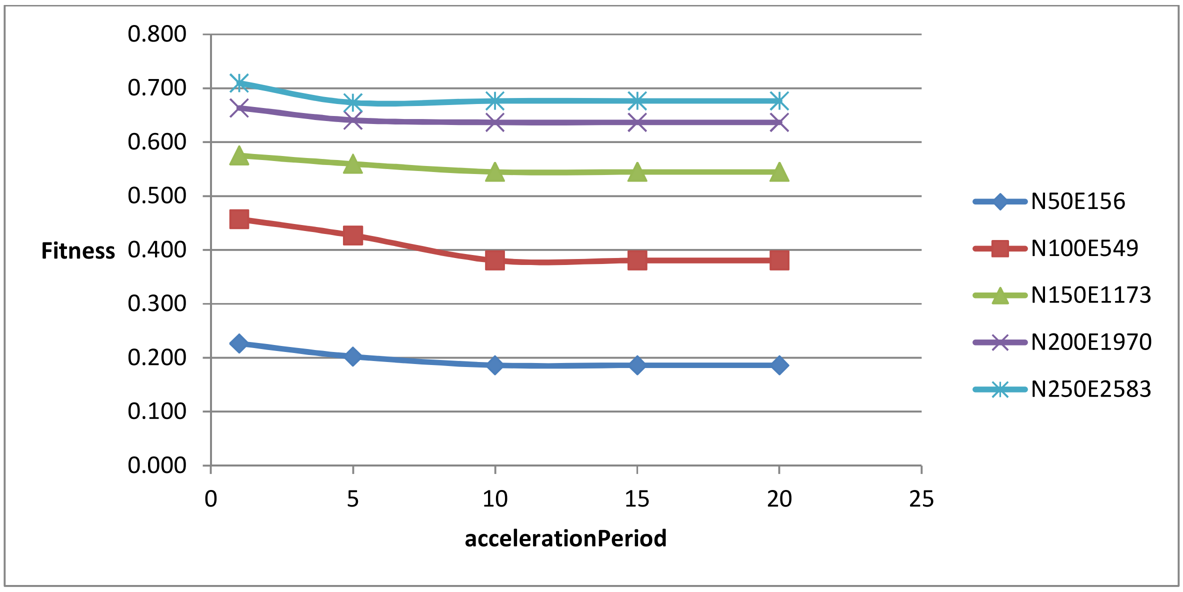


Figure 6.18 Fitness values of the layouts for the datasets in Table 6.8 when examining the values of the accelerationPeriod parameter (1st round)

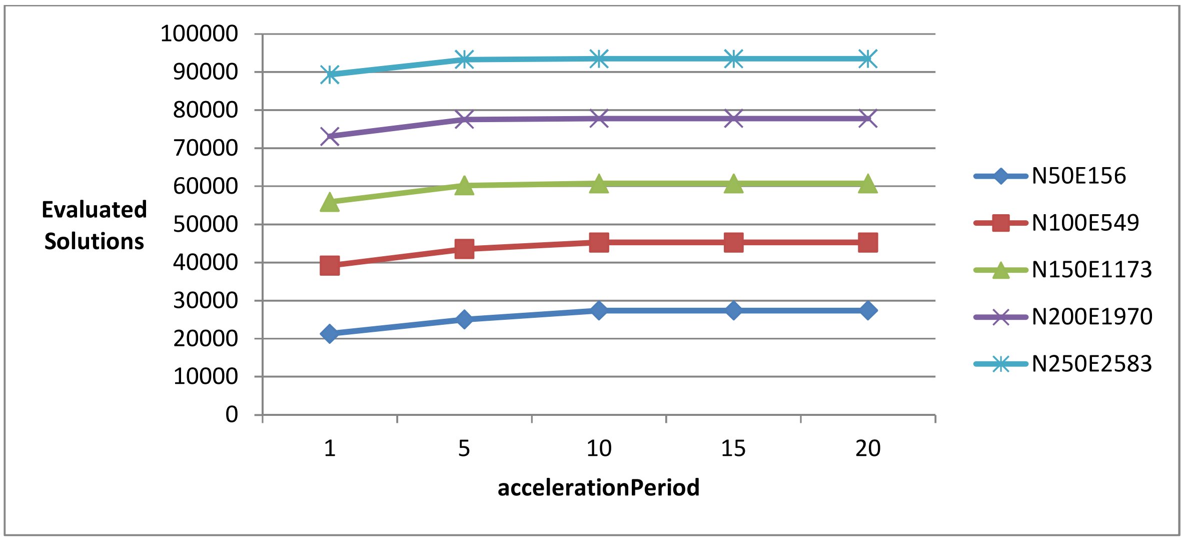


Figure 6.19 Number of solutions for drawing the layouts for the datasets in Table 6.8 when examining the values of the accelerationPeriod parameter (1st round)

To select a proper value for the *accelerationRate* parameter, we fixed the value of *accelerationPeriod* to 10, and we examined the following values for the *accelerationRate* {0.01, 0.05, 0.10, 0.15, 0.20}. The results in Figure 6.20 indicate that the value 0.01 is the one which should be selected as it generates layouts with better fitness values compared to the other values in the set. There is no need to examine the number of evaluated solutions since the values of fitness function are small with 0.01 acceleration rate. We could have tested smaller values, but our main target in this tuning process was getting a proper starting value not the final value of the parameter.

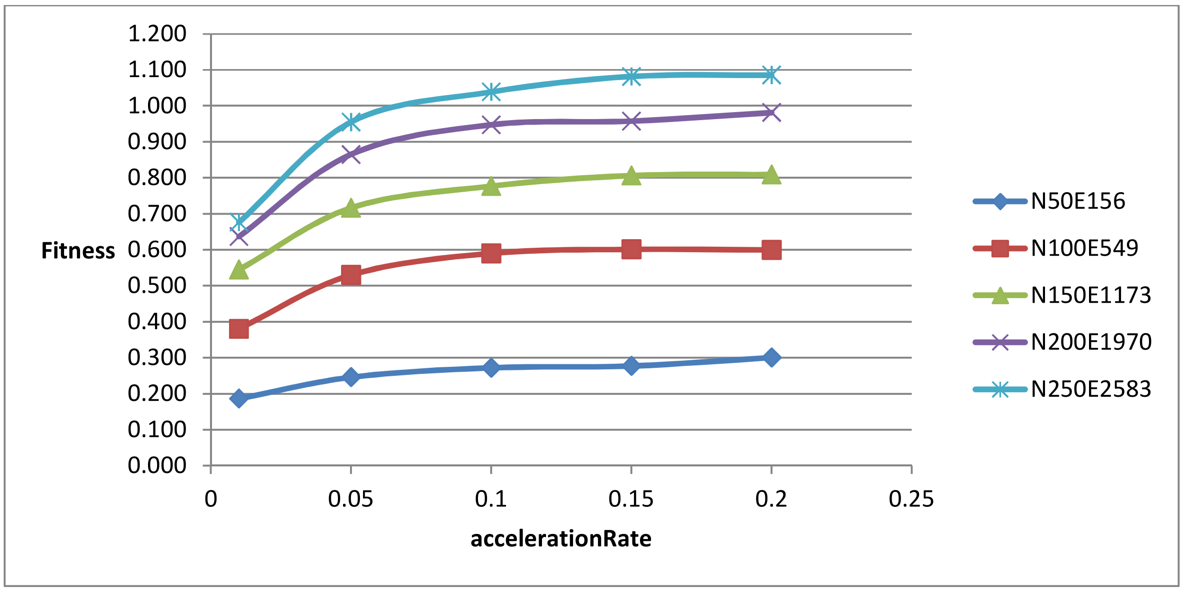


Figure 6.20 Fitness values of the layouts for the datasets in Table 6.8 when examining the values of the accelerationRate parameter (1st round)

We performed another round for calibrating the values for both parameters in the same manner which we followed in the first round but we used the values which we got in the first round as starting values. The set of values used for *accelerationPeriod* in the second round was {6, 7, 8, 9, 10}. The fitness values were close to each other as shown in Figure 6.21, with an advantage to the value 9 in some graph layouts (graphs with group label N100E549 for example). So, we picked that value for *accelerationPeriod* and we examined the following set of values for *accelerationRate* {0.01, 0.02, 0.03, 0.04, 0.05}. Again, we selected the value 0.01 with reference to the results demonstrated in Figure 6.22.

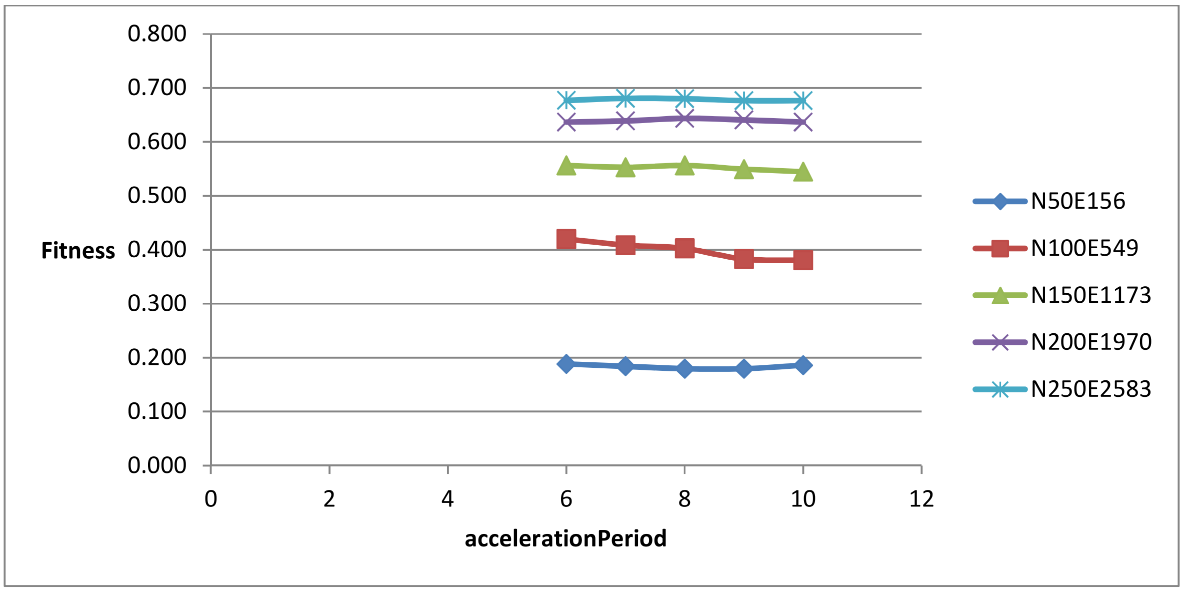


Figure 6.21 Fitness values of the layouts for the datasets in Table 6.8 when examining the values of the accelerationPeriod parameter (2nd round)

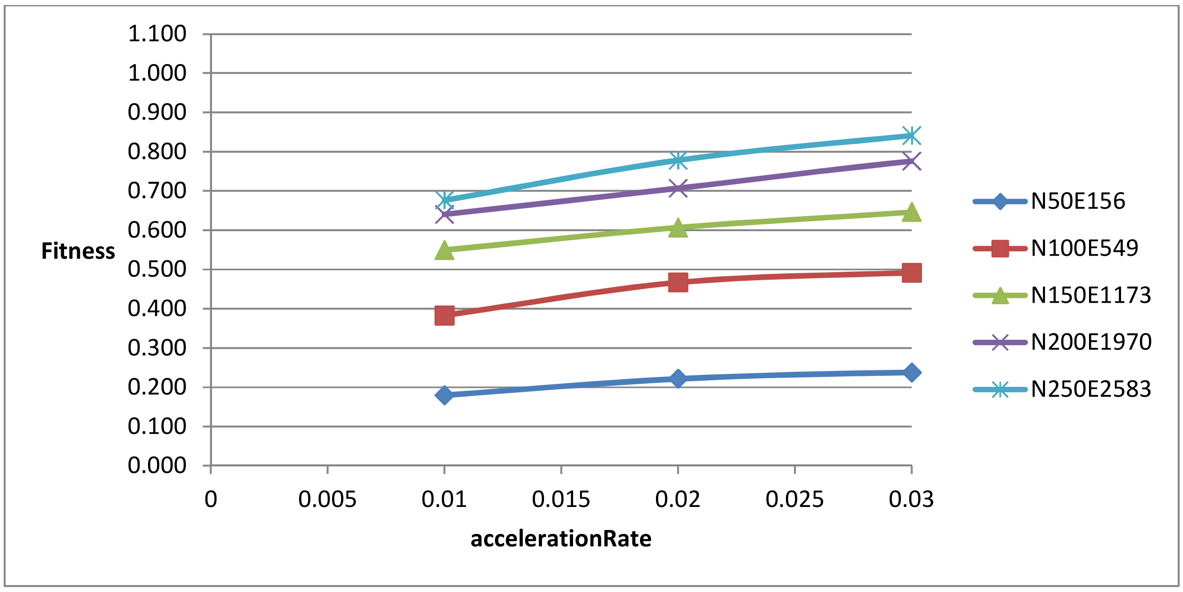


Figure 6.22 Fitness values of the layouts for the datasets in Table 6.8 when examining the values of the accelerationRate parameter (2nd round)

After selecting reasonable values for the newly introduced parameters, we implemented a comparison between the path relinking procedure with a fixed (constant) step-size for moving along the path, and the same procedure but with a variable step-size. We applied both strategies on four groups of randomly generated connected graph layouts, based on Erdos-Renyi model, with a different number of nodes and edges as shown in Table 6.9.

Table 6.9 Characteristics of the graph datasets used in the comparison between the two strategies for moving along the path

Graph Set	Nodes	Edges	Density
1	150	1229	0.11
2	200	1990	0.1
3	250	2801	0.09
4	300	3588	0.08

The two variations used the same values of all path relinking parameters except for the newly introduced parameters as they are only related to the variable step-size strategy. We ran both of them until reaching the stopping criterion. The results showed that using a variable step-size to move along the path can produce better graph layouts with a lower number of evaluated solutions than a fixed step-size strategy, as shown in Figure 6.23 and Figure 6.24.

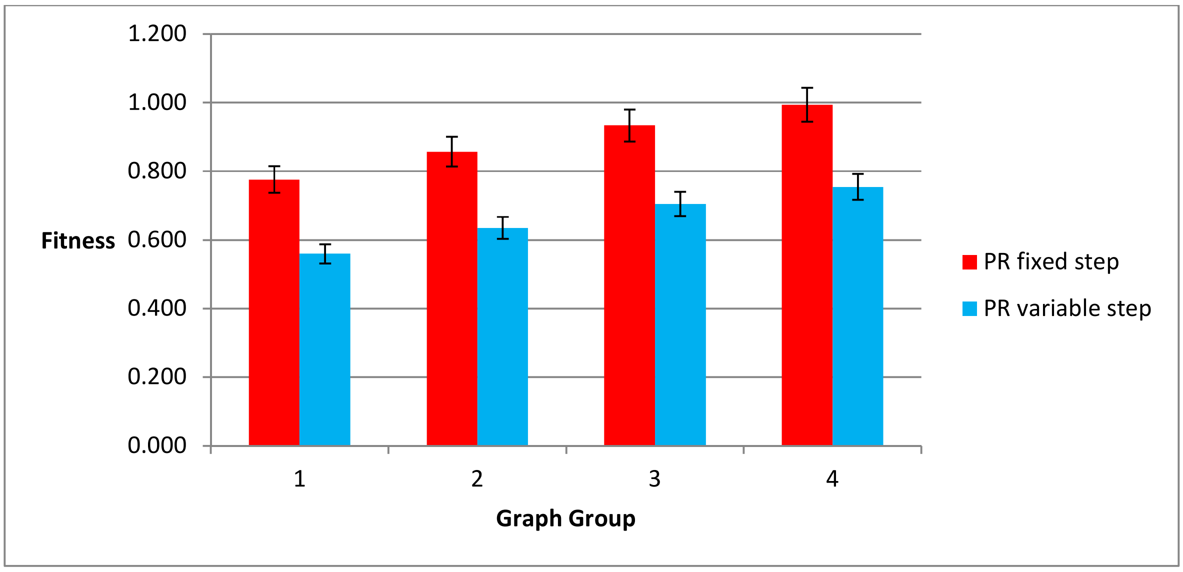


Figure 6.23 Fitness values with 95% confidence interval for the layouts of the datasets in Table 6.9 when applying the two strategies of moving along the path

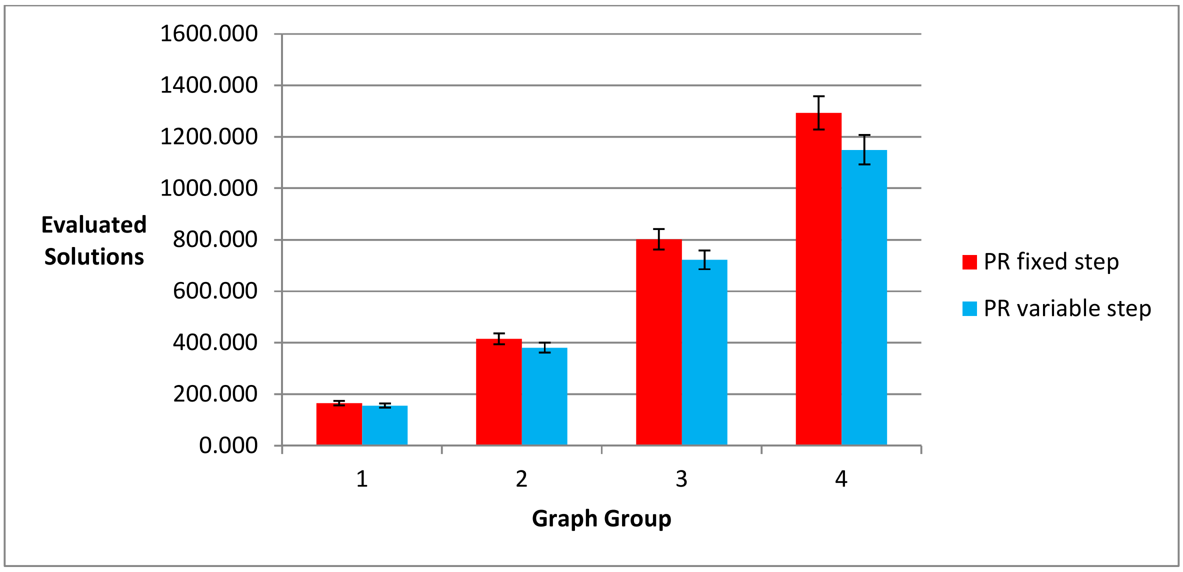


Figure 6.24 Number of solutions with 95% confidence interval for the layout of the graph datasets in Table 6.9 when applying the two strategies of moving along the path

Our improved path relinking procedure will add some changes to Algorithm 6.2 and Algorithm 6.3 that were discussed in Section 6.2.1. The two parameters, *accelerationPeriod* and *accelerationRate*, will be introduced in the *PathRelinking()* and *MoveAlongPath()* procedures, as shown in Algorithm 6.4, where the two parameters have been added to the list of parameters of the *MoveAlongPath()* procedure, and in Algorithm 6.5 (Line 5 and Line 6), where both parameters are used to intensify the searching process. Note that, the stopping conditions for moving along the path are still the same as described in Algorithm 6.3: the first is when an intermediate solution reaches the guiding solution, and the second is when the length of the path reaches a predefined value of a maximum length.

Since the results of the experiment show that the variable step-size strategy used in moving along the path is better than the fixed step-size strategy, we performed an intensive parameter tuning on all the parameters of our improved path relinking procedure, as will be described in the next section, in order to get a solid graph drawing algorithm that can be compared with simulated annealing and tabu search graph drawing algorithms.

Given:

`PRmaxIterations`: predefined value of the number of iterations to repeat the path relinking procedure.

`pathSqrSize`: predefined square size where path relinking candidate solutions are located on its border.

`pathLength`: predefined value representing the maximum length of the path.

`accelerationPeriod`: predefined number of iterations required for updating the searching step-size.

`accelerationRate`: predefined value representing the rate of decreasing the searching step-size.

Algorithm:

```
1: i = 0
2: while i < PRmaxIterations && Size(refSet) > 1 do
3:  SelectSourceDestination(refSet, source, destination)
/* returns source and destination selected from the reference set refSet */
4:  candidateLayout1 = MoveAlongPath(source, destination, pathLength, pathSqrSize,
      accelerationRate, accelerationPeriod)  /* forward path */
5:  candidateLayout2 = MoveAlongPath(destination, source, pathLength, pathSqrSize,
      accelerationRate, accelerationPeriod)  /* backward path */
6:  UpdateReferenceSet(refSet, Min(candidateLayout1, candidateLayout2))
7:  i = i + 1
8: end while
```

Algorithm 6.4 Improved PathRelinking() procedure

Algorithm:

```
MoveAlongPath (source, destination, pathLength, pathSqrSize, accelerationRate,
      accelerationPeriod)
1: length = 0
2: updateSquare = 0
3: while source != destination || length < pathLength
4:  for v in V do
5:   if length mod accelerationPeriod == 0 then  /* variable step-size for a path move
6:      updateSquare += accelerationRate
7:   end if
8:   position = ShortestEuclidean(source[v], destination[v], pathSqrSize + updateSquare)
9:  /* position with shortest distance around the square from the node in source to
      destination */
10:  fitness = Fitness(layout[position])
11:  move source[v] to position if position ∉ tabuSet
12:  tabuSet = tabuSet ∪ {(v, layout[position], iteration)}
13: end for
14: length = length + 1
15:end while
```

Algorithm 6.5 Improved MoveAlongPath() procedure

6.3.3 Parameter Tuning

There are six parameters which affect our improved path relinking procedure: the number of iterations to repeat the path relinking procedure (*PRmaxIterations*), the size for the maximum number of solutions that can be added to the reference set of path relinking (*refSize*), the maximum length of the path (*pathLength*), the square size where path relinking candidate solutions are located on its border (*pathSqrSize*), the number of iterations required to update the size of the square (*accelerationPeriod*), and the rate of decreasing the searching step-size (*accelerationRate*).

For tuning the values of these parameters, we applied our improved graph drawing algorithm on 100 random connected graphs which were divided into five sets such that each set had a different number of nodes and edges, as described in Table 6.10.

Table 6.10 Characteristics of the graph datasets used in tuning the parameters of our improved TS+PR graph drawing algorithm

Graph Set	Nodes	Edges	Density	Label
1	50	147	0.120	N50E147
2	100	519	0.105	N100E519
3	150	1117	0.100	N150E1117
4	200	1791	0.090	N200E1791
5	250	2490	0.080	N250E2490

Since the improved procedure is called within our tabu search drawing algorithm, we used the same values of the parameters of tabu search that we obtained in Chapter 4 (Section 4.5.2). On the other hand, in order to calibrate the values of the parameters of the improved path relinking, we followed the same incremental testing process we performed with all the other methods. The process was divided into three phases. In phase I, we selected the values according to our previous parameters testing described in our basic path relinking implementation in Section 6.2.2 and the values of the newly introduced parameters described in Section 6.3.2. In phase II of the experimental process for tuning the parameters of our improved path relinking procedure, we performed another round of further tuning similar to the process we followed in phase I using different graph datasets but with the same number of nodes and edges. In phase III, we focused on the number of evaluated solutions performed by the drawing algorithm when it reached a certain fitness value.

i. Phase I

In phase I, we selected the values according to our previous parameters testing described in our basic path relinking implementation in Section 6.2.2 and the values of the newly introduced parameters described in Section 6.3.2. The initial values of the parameters were: *PRmaxIterations* = 4, *refSize* = 20, *pathLength* = 10, *pathSqrSize* = 18, *accelerationPeriod* = 9, *accelerationRate* = 0.01. We started with one parameter, tested it thoroughly with different values, and selected the value which draws layouts with the minimum fitness value compared to the other values. If the fitness values were too close to each other, we would select the values based on the ones which performed the lowest number of evaluated solutions. We fixed the value of the first parameter and we moved on to test another parameter in the same manner, and so forth.

We started the tuning process with the *PRmaxIterations* parameter by testing the values of the set {1, 4, 7, 10}. Figure 6.25 shows that increasing the value of this parameter would minimise the value of the fitness function of the generated layout. According to the set of values which we tested, the best value to choose was 10.

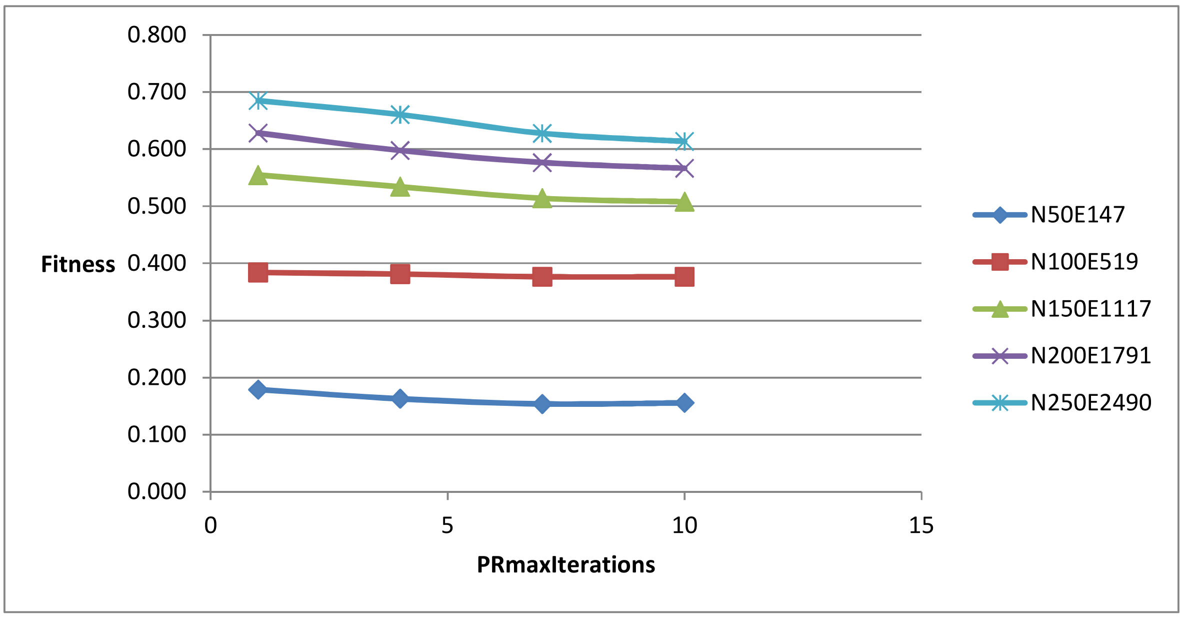


Figure 6.25 Fitness values of the improved drawing algorithm when tuning the PRmaxIterations parameter (phase I)

In the next parameter (*refSize*), we selected the set {10, 20, 30, 40} to be used in calibrating this parameter. With reference to Figure 6.26, the best value for *refSize* that gave the best fitness value was 20. Note that, all the tested values led to producing very close fitness values, but as the value of this parameter increases, it slightly increases the number of evaluated solutions, as shown in Figure 6.27. We selected the value 20, as it gave a fitness value (on the graphs with label N250E2490) that was slightly better than the other values and the number of evaluated solutions performed by the algorithm when using this value is less than the evaluated solutions when we test this parameter on the values 30 and 40).

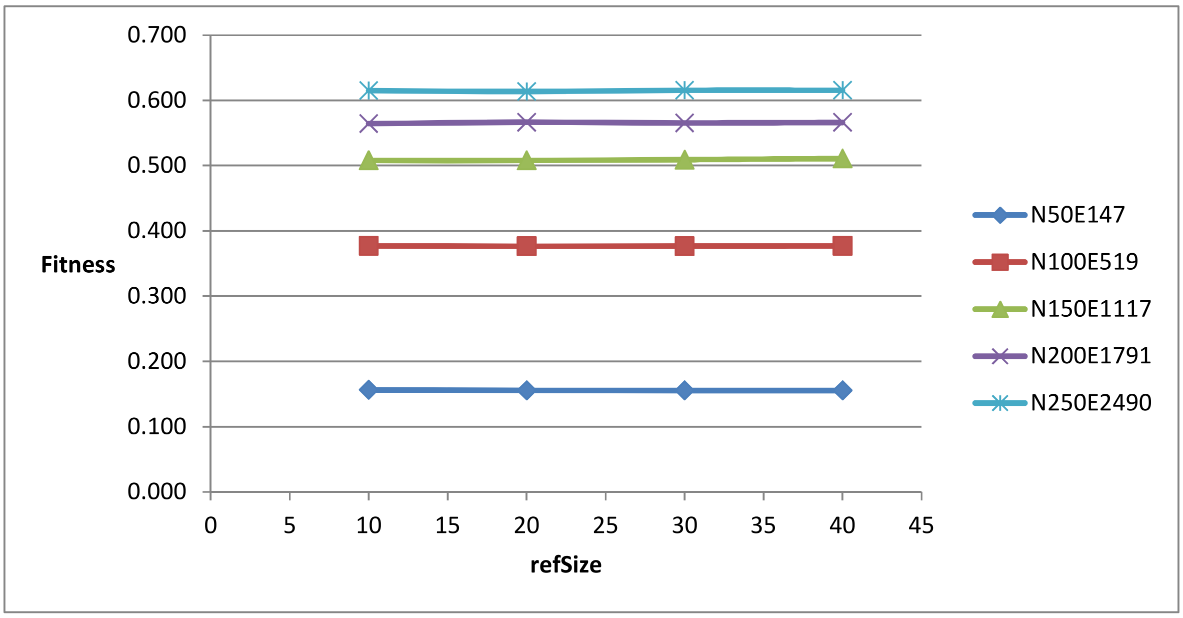


Figure 6.26 Fitness values of the improved drawing algorithm when tuning the refSize parameter (phase I)

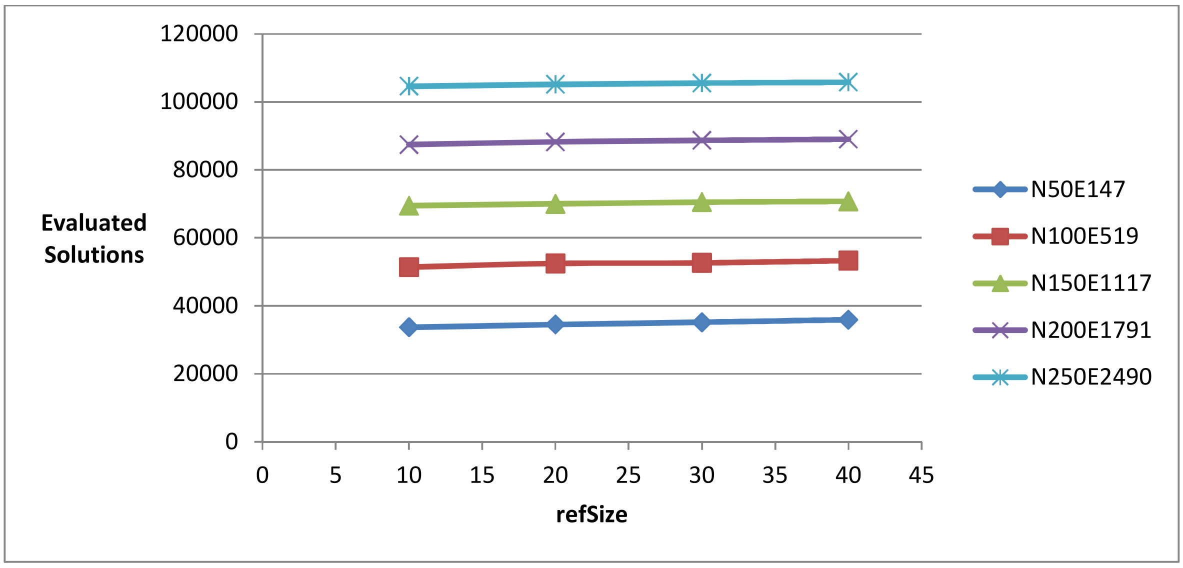


Figure 6.27 Number of evaluated solutions of the improved drawing algorithm when tuning the refSize parameter (phase I)

The length of the path from the initial solution to the target solution (*pathLength*) was tested with the following set of values: {10, 20, 30, 40}. After testing all these values, we selected the value 20. We chose this value although it did not give better fitness compared to the value 10 on small graphs, but it has the same behaviour on larger graphs as shown in Figure 6.28. We first need to test the effect of the initial square size value on longer paths. If the effect is not significant, then we could select the value 10 in phase II of parameters testing.

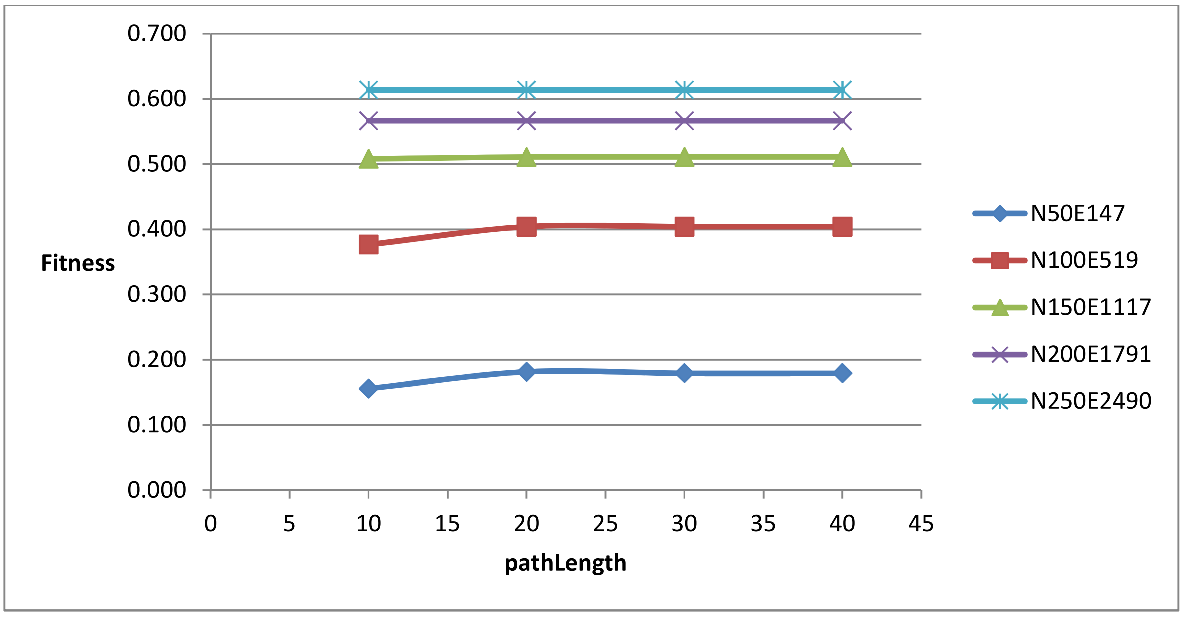


Figure 6.28 Fitness values of the improved drawing algorithm when tuning the pathLength parameter (phase I)

The *pathSqrSize* parameter was tested with the values {5, 10, 15, 20}. According to Figure 6.29, the best value that could be picked is 20 since the fitness value was slightly smaller as the graph size became larger. The value 15 also produced good results but when applied on larger graphs, the value 20 was better.

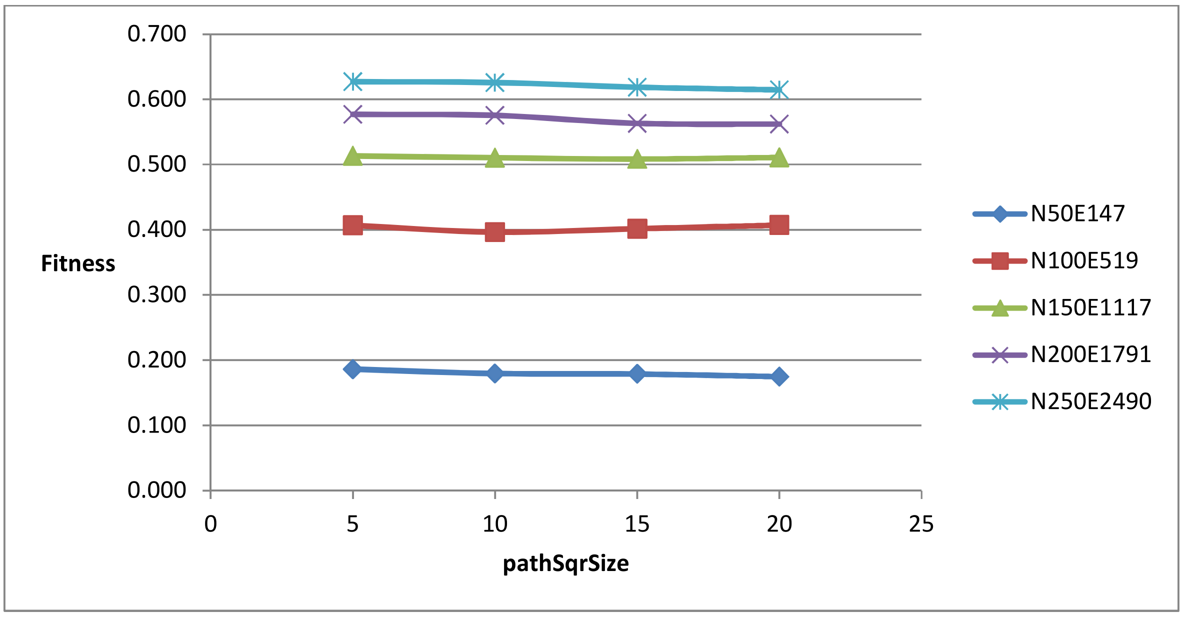


Figure 6.29 Fitness values of the improved drawing algorithm when tuning the pathSqrSize parameter (phase I)

To test the effect of the *accelerationPeriod* parameter, we tested it with the following values: {1, 5, 9, 13}. Figure 6.30 shows that changing the value of this parameter did not greatly affect the value of the fitness function. But Figure 6.31 shows that increasing the value of this parameter would slightly increase the number of evaluated solutions. That is why we chose the value 5 although there was no big difference with the fitness values produced when *accelerationPeriod* was set to 9 or 13, but it was better on larger graphs with a lower number of evaluated solutions.

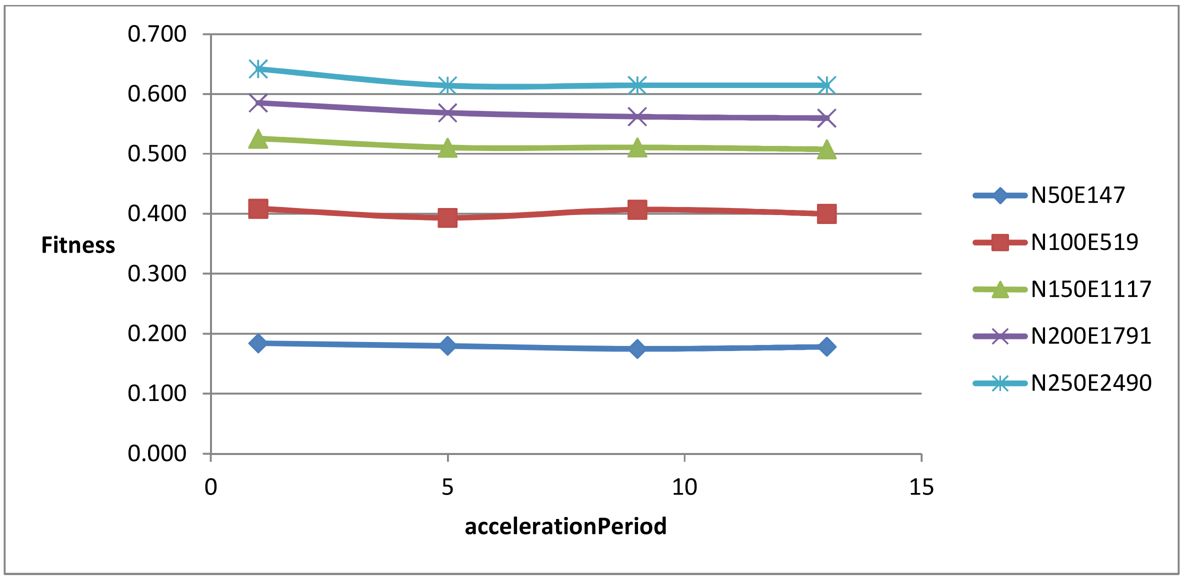


Figure 6.30 Fitness values of the improved drawing algorithm when tuning the accelerationPeriod parameter (phase I)

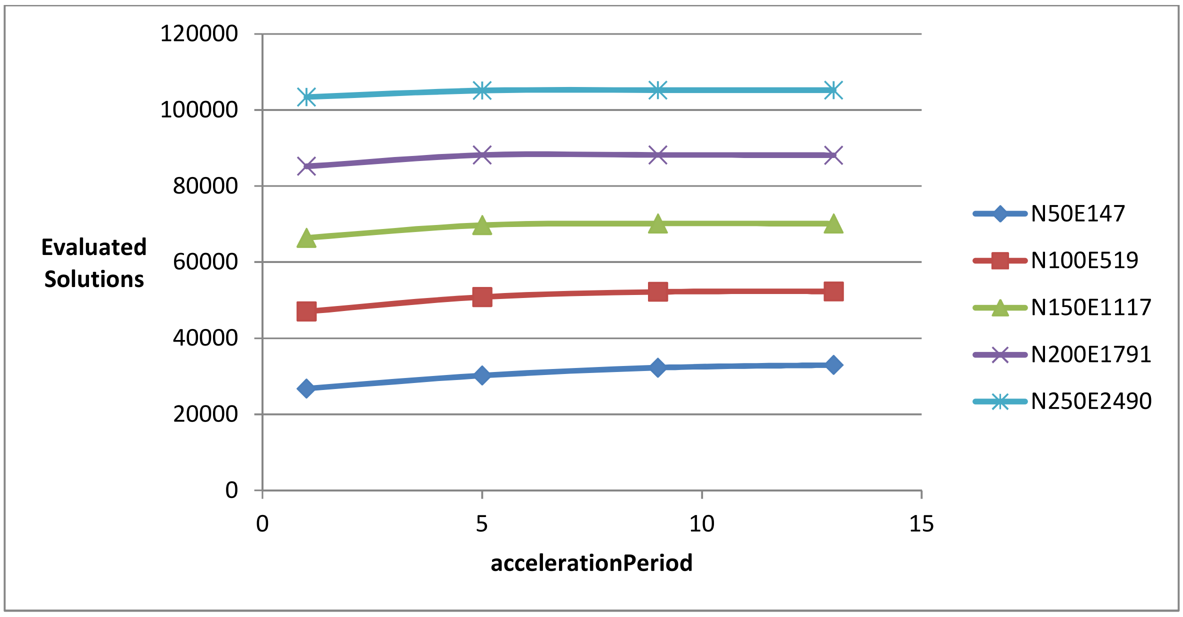


Figure 6.31 Number of evaluated solutions of the improved drawing algorithm when tuning the accelerationPeriod parameter (phase I)

The last parameter that was tested in phase I was *accelerationRate* which was tested with the values {0, 0.05, 0.1, 0.15}. Increasing the value of this parameter resulted in an increase in the value of the fitness function as shown in Figure 6.32 when the values went beyond the value 0.05. On the other hand, setting the value 0 to this parameter had produced larger fitness values compared to those when the value 0.05 was assigned to this parameter. Therefore, we chose the value 0.05 in this phase, but in the next phase, we will test the value of this parameter with a set of values in the range between 0 and 0.05 to examine the behaviour of the fitness function in that specific range.

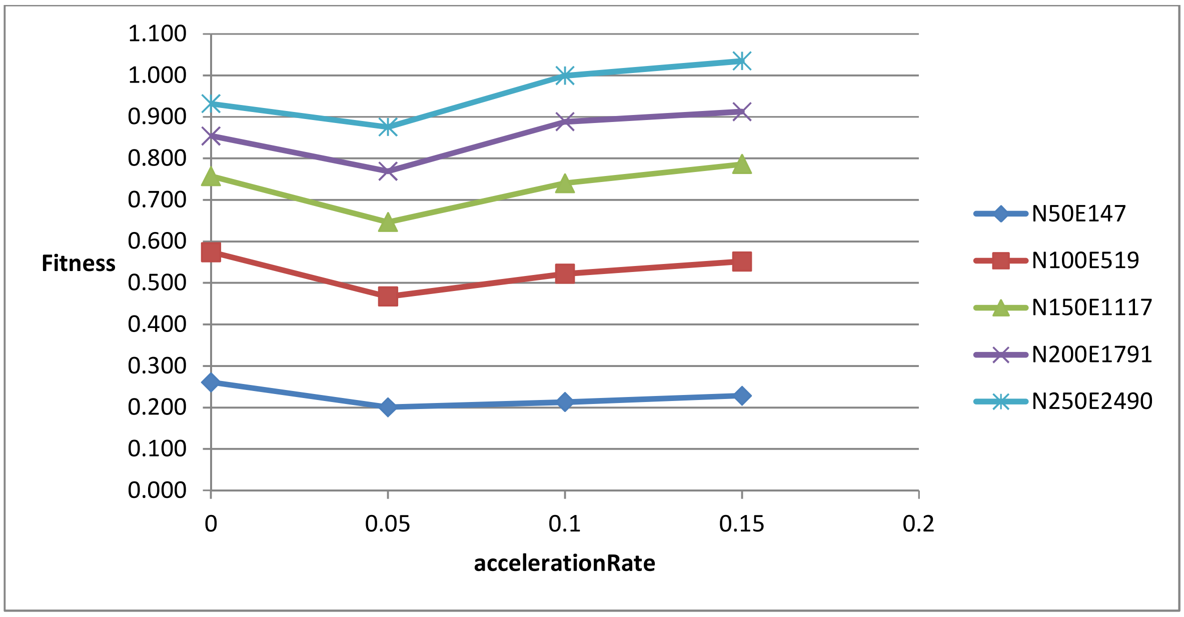


Figure 6.32 Fitness values of the improved drawing algorithm when tuning the accelerationRate parameter (phase I)

ii. Phase II

We performed another round of further tuning similar to the process we followed in phase I using different graph datasets but with the same number of nodes and edges, as described earlier in Table 6.10. In this phase, all the parameters started with the values which were chosen in the first phase. When a parameter was tested, a set of values that are close to the value that was chosen in the previous phase was selected. The behaviour of the drawing algorithm was similar to its behaviour in the previous phase for all the parameters with a slight change in the selected values of the parameters since the set of values was different in this phase. The parameters were assigned the following values at the end of this phase: *PRmaxIterations* = 10, *refSize* = 20, *pathLength* = 15, *pathSqrSize* = 20, *accelerationPeriod* = 5, *accelerationRate* = 0.0025.

iii. Phase III

We selected the values of the parameters that made the algorithm implement the lowest number of evaluated solutions. We assumed that a good-enough graph layout is a layout in which its fitness value is slightly larger than the best fitness value produced in the experiments of the previous phase as we explained earlier in Chapter 4 (Section 4.3.2). We set a target fitness value for each test case (from phase II with an increase of 12.5%) and we tested number of evaluated solutions required to reach that value. The value of the parameter that gave the minimum number of evaluated solutions was chosen. Different graph datasets (different layouts) were randomly generated and used in this phase as well, but with the same number of nodes and edges of the previous two phases.

The initial values of the parameters in this phase were the final values which were chosen from the previous phase. We started with the *PRmaxIterations* parameter by testing a set of values {2, 4, 6, 8, 10}. Although the value 2 would make the algorithm perform the lowest number of evaluated solutions as shown in Figure 6.33, it did not reach the target fitness on the small graph layouts according to the highlighted cell in Table 6.11. Thus, we picked the value 4 since the algorithm had reached the target fitness value with all graph layouts and it had performed the lowest number of evaluated solutions.

Table 6.11 Fitness values reaching a target value by the improved drawing algorithm when tuning the PRmaxIterations parameter (phase III)

	Fitness				
PRmaxIterations	**N50E147**	**N100E519**	**N150E1117**	**N200E1791**	**N250E2490**
2	0.180	0.372	0.483	0.574	0.644
4	0.164	0.376	0.484	0.574	0.644
6	0.161	0.377	0.484	0.574	0.645
8	0.163	0.379	0.483	0.572	0.643
10	0.161	0.377	0.483	0.573	0.642
Target	**0.164**	**0.382**	**0.487**	**0.581**	**0.650**

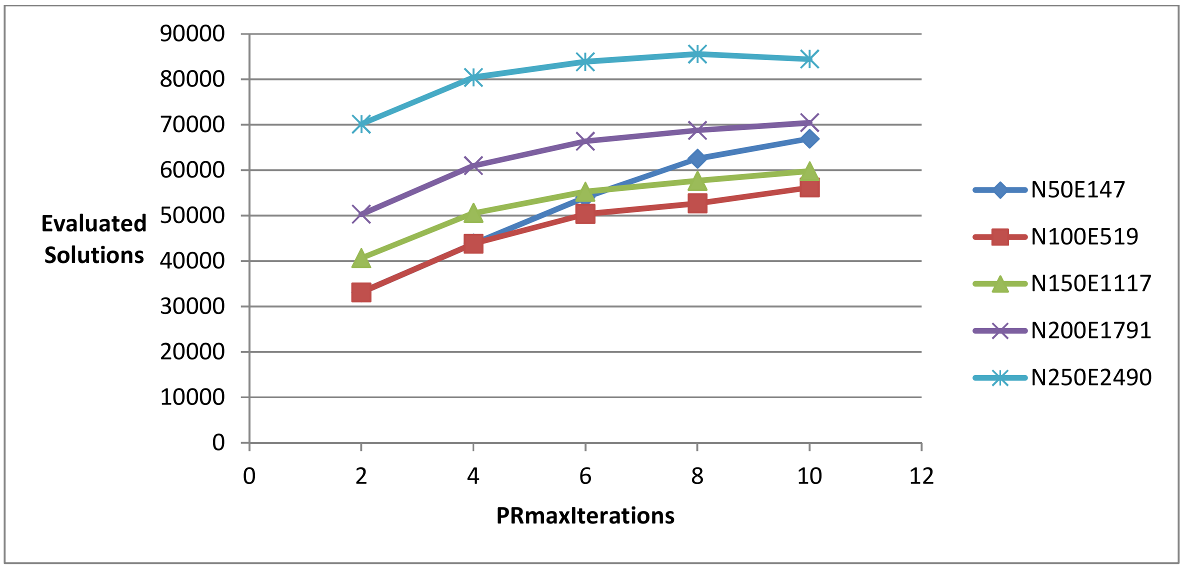


Figure 6.33 Number of evaluated solutions of the improved drawing algorithm when tuning the PRmaxIterations parameter (phase III)

For the size of candidate elite solutions in the reference set, *refSize*, we examined the following set of values: {12, 16, 20, 24}. The highlighted cells in Table 6.12 show that the algorithm could not reach the target fitness value when *refSize* had the values 16 or 24. With reference to Figure 6.34, we selected the value 20 since the algorithm had performed a slightly lower number of evaluated solutions compared to the performance of the algorithm when the value 12 was assigned to the *refSize* parameter.

Table 6.12 Fitness values reaching a target value by the improved drawing algorithm when tuning the refSize parameter (phase III)

	Fitness				
refSize	**N50E147**	**N100E519**	**N150E1117**	**N200E1791**	**N250E2490**
12	0.164	0.375	0.484	0.573	0.644
16	0.165	0.376	0.484	0.574	0.644
20	0.164	0.376	0.484	0.574	0.644
24	0.165	0.376	0.484	0.574	0.644
Target	**0.164**	**0.382**	**0.487**	**0.581**	**0.650**

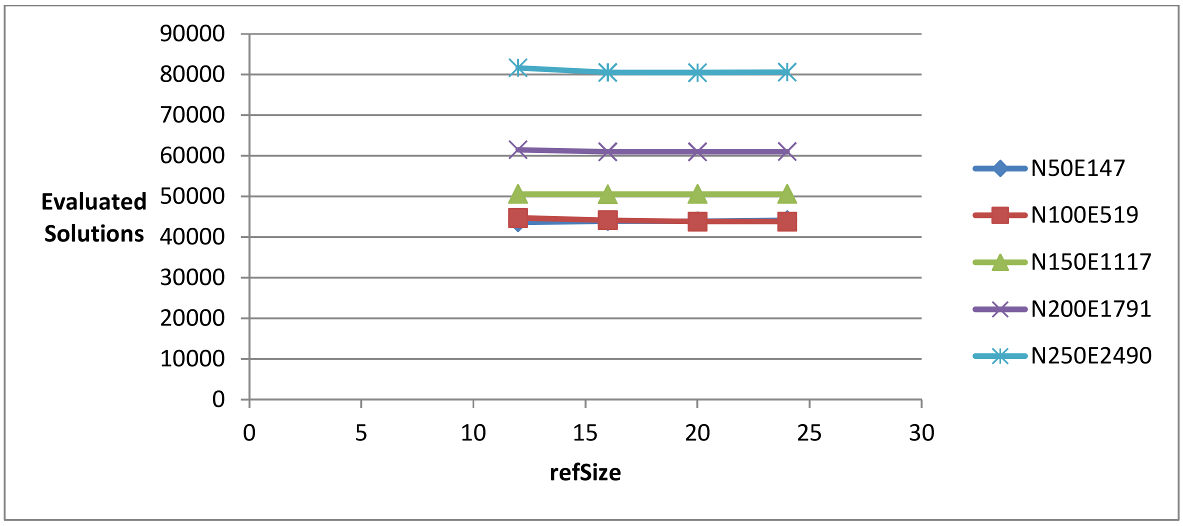


Figure 6.34 Number of evaluated solutions of the improved drawing algorithm when tuning the refSize parameter (phase III)

The third parameter, *pathLength*, was tested with the values {11, 15, 19, 23}. None of these values (except the value 15) could help the algorithm in reaching the target fitness values for all the graph layouts as shown in the highlighted cells of Table 6.13. Therefore, we selected the value 15 since it was the only one that reached the target fitness value for all the test cases.

Table 6.13 Fitness values reaching a target value by the improved drawing algorithm when tuning the pathLength parameter (phase III)

	Fitness				
pathLength	**N50E147**	**N100E519**	**N150E1117**	**N200E1791**	**N250E2490**
11	0.242	0.367	0.471	0.568	0.639
15	0.164	0.376	0.484	0.574	0.644
19	0.167	0.373	0.511	0.579	0.644
23	0.172	0.376	0.516	0.579	0.644
Target	**0.164**	**0.382**	**0.487**	**0.581**	**0.65**

Two out of the four values {12, 16, 20, 24} which we used for tuning the fourth parameter, *pathSqrSize*, led to the failure of the algorithm to reach the target fitness value as shown in the highlighted cells of Table 6.14. With reference to Figure 6.35, we chose the value 20 although the values 12 and 16 gave a lower number of evaluated solutions on small graph layouts, but that was not the case with large graph layouts which is more important to us in this phase since we are trying to improve the performance of the drawing algorithm. On the other hand, the value 24 did not reach the target fitness value on small graph layouts.

Table 6.14 Fitness values reaching a target value by the improved drawing algorithm when tuning the pathSqrSize parameter (phase III)

	Fitness				
pathSqrSize	**N50E147**	**N100E519**	**N150E1117**	**N200E1791**	**N250E2490**
12	0.165	0.376	0.483	0.577	0.644
16	0.164	0.377	0.480	0.576	0.643
20	0.164	0.376	0.484	0.574	0.644
24	0.170	0.377	0.480	0.573	0.644

Target	**0.164**	**0.382**	**0.487**	**0.581**	**0.65**

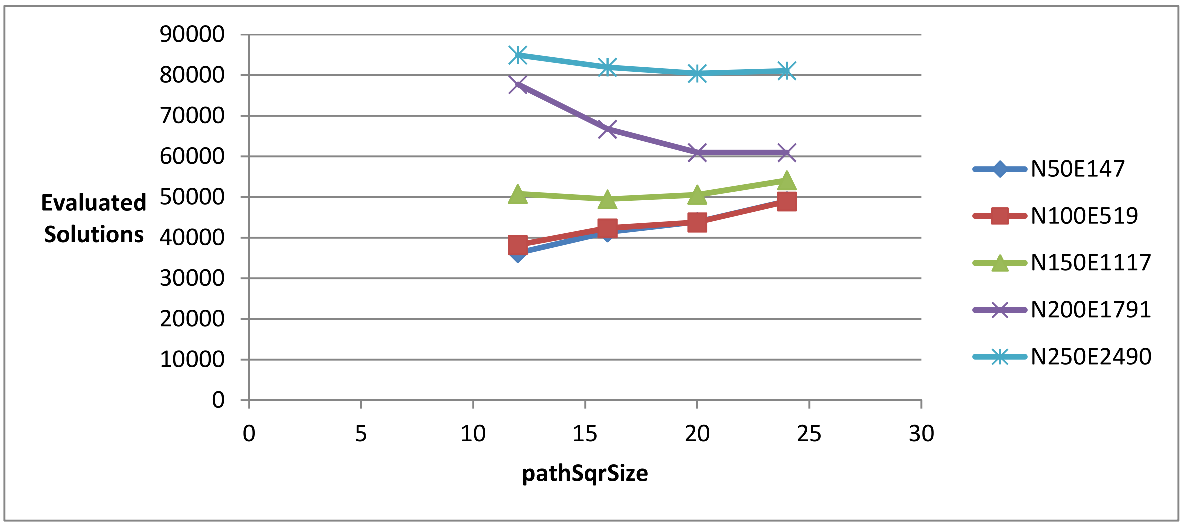


Figure 6.35 Number of evaluated solutions of the improved drawing algorithm when tuning the pathSqrSize parameter (phase III)

During the tuning process of the fifth parameter, *accelerationPeriod*, the behaviour of the algorithm was not clear when we chose a set of four values only for tuning the parameter. Thus, we increased the number of values in order to examine the behaviour of the algorithm and to get a proper indication of its performance. The following values were tested: {2, 3, 4, 5, 6, 7, 8, 9, 10}.

After testing all the values, we selected the value 7 to be assigned to the *accelerationPeriod* parameter. This value did not reach the target fitness on the smallest graph dataset as shown in Table 6.15, but as Figure 6.36 indicates, the number of evaluated solutions had greatly dropped down with value 7. Since our main target was to speed up the drawing process while generating good graph layouts, we chose the value 7 because there was no big difference in the number of evaluated solutions as the size of the graph layouts increased. On the other hand, the algorithm had generated graph layouts with fitness values which reach the target fitness values for all graphs when the values 4 and 5 were assigned to the *accelerationPeriod* parameter, but the number of evaluated solutions was very large when the graph size increased.

Table 6.15 Fitness values reaching a target value by the improved drawing algorithm when tuning the accelerationPeriod parameter (phase III)

	Fitness				
accelerationPeriod	**N50E147**	**N100E519**	**N150E1117**	**N200E1791**	**N250E2490**
2	0.185	0.403	0.511	0.580	0.644
3	0.168	0.381	0.507	0.579	0.645
4	0.164	0.375	0.483	0.579	0.645
5	0.164	0.376	0.484	0.574	0.644
6	0.172	0.375	0.482	0.569	0.640
7	0.205	0.373	0.479	0.568	0.641
8	0.245	0.372	0.479	0.566	0.640
9	0.254	0.372	0.474	0.570	0.641
10	0.255	0.368	0.477	0.569	0.640
Target	**0.164**	**0.382**	**0.487**	**0.581**	**0.65**

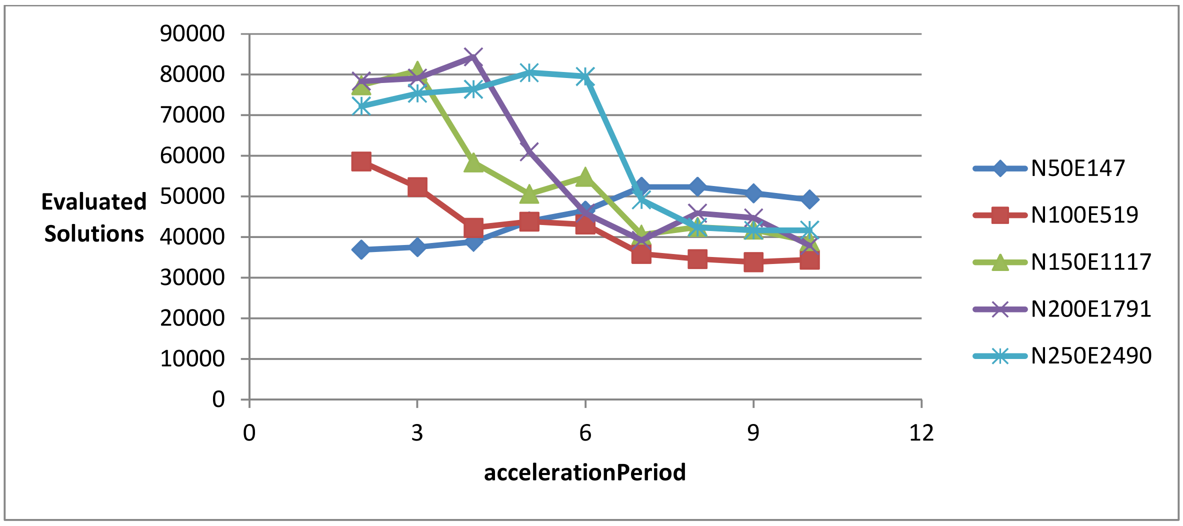


Figure 6.36 Number of evaluated solutions of the improved drawing algorithm when tuning the accelerationPeriod parameter (phase III)

With the last parameter, *accelerationRate*, we had a similar situation to the one we had in the previous parameter, where the behaviour of the algorithm was not very clear with the four values we initially chose. Therefore, we tested this parameter with many values {0.001, 0.0015, 0.002, 0.0025, 0.003, 0.0035, 0.004, 0.0045, 0.005}. After testing all these values, there were two values which could be selected for this parameter: 0.002 or 0.0025. When the parameter was assigned any of these two values, the number of evaluated solutions performed by the algorithm was minimum compared to the other values. Although the target fitness value was not reached when those two values were selected on the smallest graph layouts only, but that was the case also when all the other values were tested as shown in the highlighted cells of Table 6.16. Since we were looking to generate good graph layouts besides speeding up the algorithm to lay out larger graphs, we chose the value 0.002 as the number of evaluated solutions was the lowest, in most of the test cases, compared to all other values as shown in Figure 6.37.

Table 6.16 Fitness values reaching a target value by the improved drawing algorithm when tuning the accelerationRate parameter (phase III)

	Fitness				
accelerationRate	**N50E147**	**N100E519**	**N150E1117**	**N200E1791**	**N250E2490**
0.001	0.255	0.384	0.473	0.557	0.644
0.0015	0.258	0.359	0.453	0.555	0.643
0.002	0.250	0.370	0.478	0.562	0.640
0.0025	0.205	0.373	0.479	0.568	0.641
0.003	0.168	0.374	0.480	0.569	0.642
0.0035	0.165	0.377	0.480	0.573	0.644
0.004	0.165	0.378	0.485	0.576	0.643
0.0045	0.164	0.383	0.494	0.580	0.644
0.005	0.165	0.382	0.499	0.584	0.645

Target	**0.164**	**0.382**	**0.487**	**0.581**	**0.65**

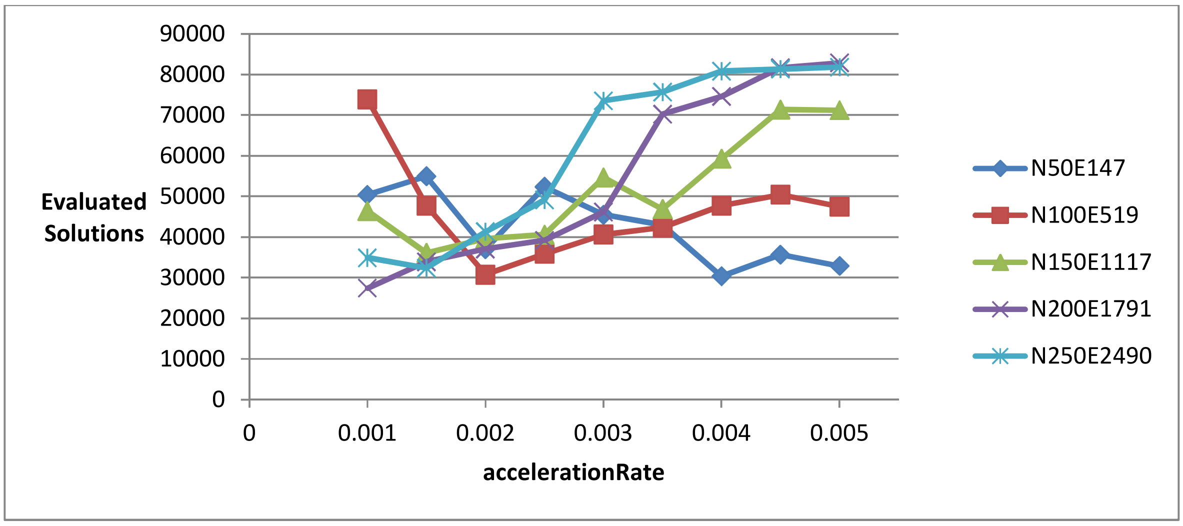


Figure 6.37 Number of evaluated solutions of the improved drawing algorithm when tuning the accelerationRate parameter (phase III)

After finishing the tuning process for all the parameters of the improved path relinking procedure, we list the value of each parameter which will be used in our coming experiments:

PRmaxIterations = 4

refSize = 20

pathLength = 15

pathSqrSize = 20

accelerationPeriod = 7

accelerationRate = 0.002

6.4 Summary

Path relinking is a relatively new neighbourhood search-based method that has proved its effectiveness in many multi-criteria optimisation problems especially when coupled with other search-based methods as an intensification step. In this chapter, we described our basic graph drawing algorithm which was based on a coupling of tabu search and path relinking. The desire to tunnel through blocked off areas created by tabu search solutions was the main reason for choosing the path relinking procedure to couple with tabu search to intensify the searching process between an initial and a guiding solutions selected from a set of elite solutions generated by the tabu search drawing algorithm. A first round of parameter tuning was performed to calibrate the values of the basic parameters of the path relinking procedure. Two improvements were proposed and applied on the path relinking procedure: a proper selection of the initial and the guiding solutions from the reference set of elite solutions, and an improved neighbourhood searching strategy based on a variable step size. The proposed improvements introduced two new parameters that could affect the performance of the procedure. Therefore, we performed a final round of the parameters calibration process in order to assign reasonable values for each parameter before we examine the performance of our improved neighbourhood search-based method compared to other neighbourhood search methods.

In the next chapter, we conduct an experiment that consists of three phases, with the aim to perform a comprehensive comparison, in terms of the speed of the drawing algorithm and the quality of the generated layouts, between three neighbourhood search-based methods: simulated annealing, tabu search, and our improved coupling of tabu search with the path relinking procedure.

Chapter 7 Experimental Results for Comparing Tabu Search with Path Relinking Versus Simulated Annealing

This chapter demonstrates the effect of coupling the tabu search graph drawing algorithm with path relinking. A comparison with simulated annealing is made by applying the methods on random and real world graph datasets. It also illustrates the process we followed for analysing the performance of our method and for testing its scalability. The experimental results are presented along with our comments and conclusions.

7.1 Introduction

In the previous experiment that was described in Chapter 5, we concluded that simulated annealing and tabu search graph drawing algorithms can generate graph layouts with better fitness values compared to the ones generated by hill climbing. In this chapter, we want to test the effect of coupling path relinking with our tabu search algorithm. In this experiment, in spite of using new randomly generated datasets, we exclude hill climbing for two reasons:

- The results of the previous experiment showed that hill climbing performed considerably worse than both tabu search and simulated annealing in all phases when being applied on random datasets and real world datasets;
- One of the main drawbacks of hill climbing is getting trapped in local optima, unlike tabu search which does not run out of solutions (as we described earlier in Chapter 4). That behaviour of hill climbing conflicts with the fact that building a reference set for path relinking requires diversity in the elite solutions generated by the other search algorithm in the pre-processing step.

Here, we need to answer the following question: ‘Does coupling the tabu search method with path relinking improve the performance of the tabu search graph drawing method?’ To answer this question we had to implement and evaluate our improved method against simulated annealing and pure tabu search graph drawing algorithms. We use the same system specifications and the same three phases of evaluations that were implemented in our previous experiment, that include: finding the best layout that can be achieved (phase I); how long it takes to generate a layout to a particular level of quality (phase II); and how good the quality of the layout is after a fixed number of evaluated solutions (phase III). The values of the parameters for each method were assigned according to the selections we made after the tuning process, as described earlier in Chapter 4 and Chapter 6.

In order to avoid overfitting, where the drawing algorithm could be tailored to the dataset used in the first experiment, we generated new random graph datasets in this experiment, based on Erdos-Renyi model that were also divided into two categories, using the same procedure we followed for generating random graphs in our previous comparison.

In the first category, we had 80 random graphs split into 4 groups of 20 test cases. All the graphs in this category had 160 nodes, randomly positioned. Each group had a different number of edges so that the density varied. The graphs in each group had the same number of nodes and edges but with different random layouts. See Table 7.1 for the characteristics of the graphs in the first category. The graphs of the second category were generated in the same way as those graphs of category II described in the previous experiment. See Table 7.2 for the characteristics of the graphs in the second category.

Table 7.1 Characteristics of the graphs in the 1st category used in comparing PR+TS, TS, and SA

Graph Set	Nodes	Edges	Density
1C	160	572	0.045
2C	160	1208	0.095
3C	160	1844	0.145
4C	160	2480	0.195

Table 7.2 Characteristics of the graphs in the 2nd category used in comparing PR+TS, TS, and SA

Graph Set	Nodes	Edges	Density
1D	60	221	0.125
2D	110	659	0.110
3D	160	1272	0.100
4D	210	2139	0.0975

7.2 Experiments on Random Graph Datasets

In a similar scenario to the experiments which were conducted in Chapter 5, we divided our experiment into three phases. The first phase focuses on the overall performance for each method where all the methods run until they completely finish execution; the second phase evaluates the speed of each algorithm when it runs for a particular level of quality; and the third phase investigates the quality of the drawn layouts after a fixed number of evaluated solutions.

The experiment includes two deterministic methods (pure tabu search and tabu search coupled with path relinking) and one stochastic method (simulated annealing). When the deterministic methods were applied on several test cases from a group of graphs with similar characteristics but with different initial layouts, we computed the average for the fitness values and the average of the number of evaluated solutions for each group of graph layouts to use in our comparison. But that was not the case with simulated annealing as it is a stochastic method. For each test case, we ran a simulated annealing graph drawing method on the same initial layout 30 times, then, we computed the median. The average of medians was calculated for the 30 runs of each test case and was compared to the results obtained after applying the deterministic methods.

7.2.1 Phase I

The three methods were applied on the datasets described in Section 7.1. In this phase, we tested the overall performance of each method by running each method until it finishes regardless of how long it took to execute. Figure 7.1 and Figure 7.2 show the values of the fitness function when the three methods were applied on the graph datasets of the first and the second categories respectively, whereas Figure 7.3 and Figure 7.4 show the number of evaluated solutions performed by each method when applied on the same datasets.

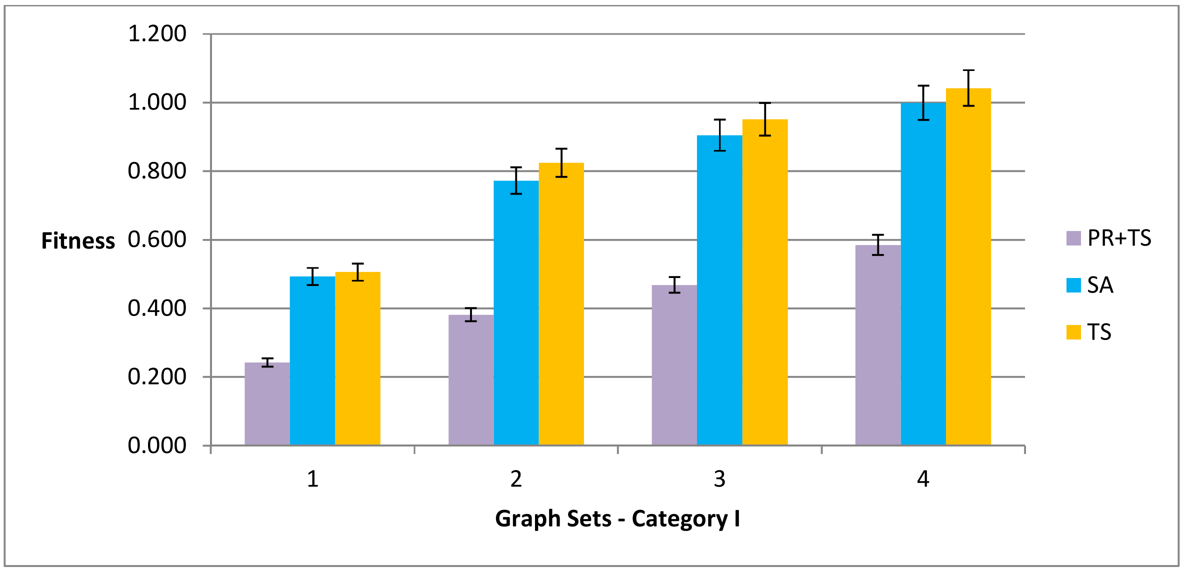


Figure 7.1 Bar chart with 95% confidence interval of the fitness function obtained by TS, SA, PR+TS when applied on the graphs of the 1st category (phase I)

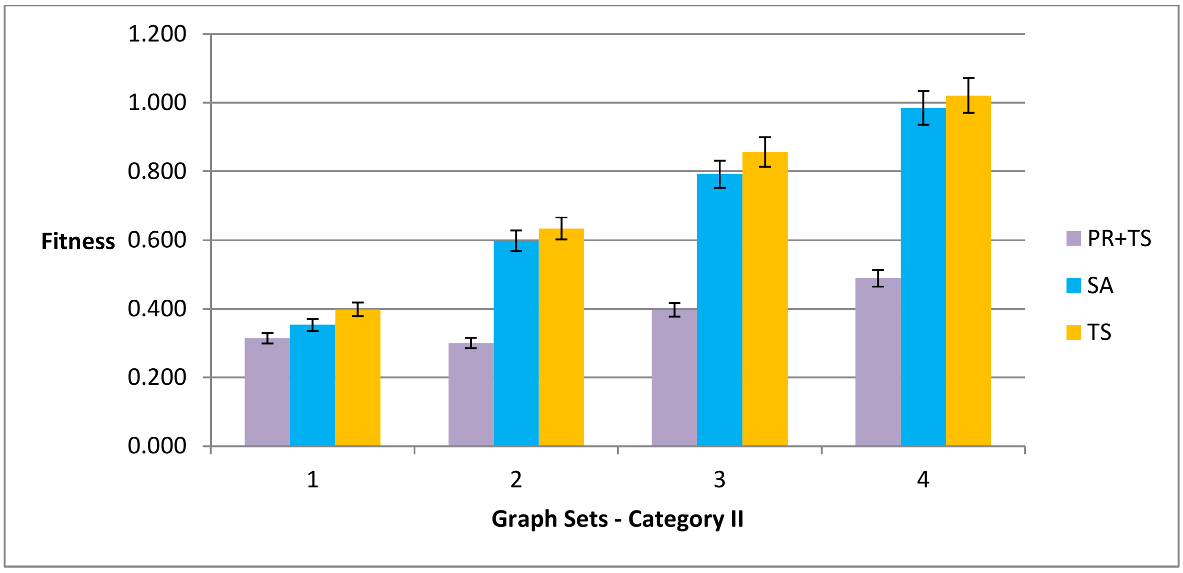


Figure 7.2 Bar chart with 95% confidence interval of the fitness function obtained by TS, SA, PR+TS when applied on the graphs of the 2nd category (phase I)

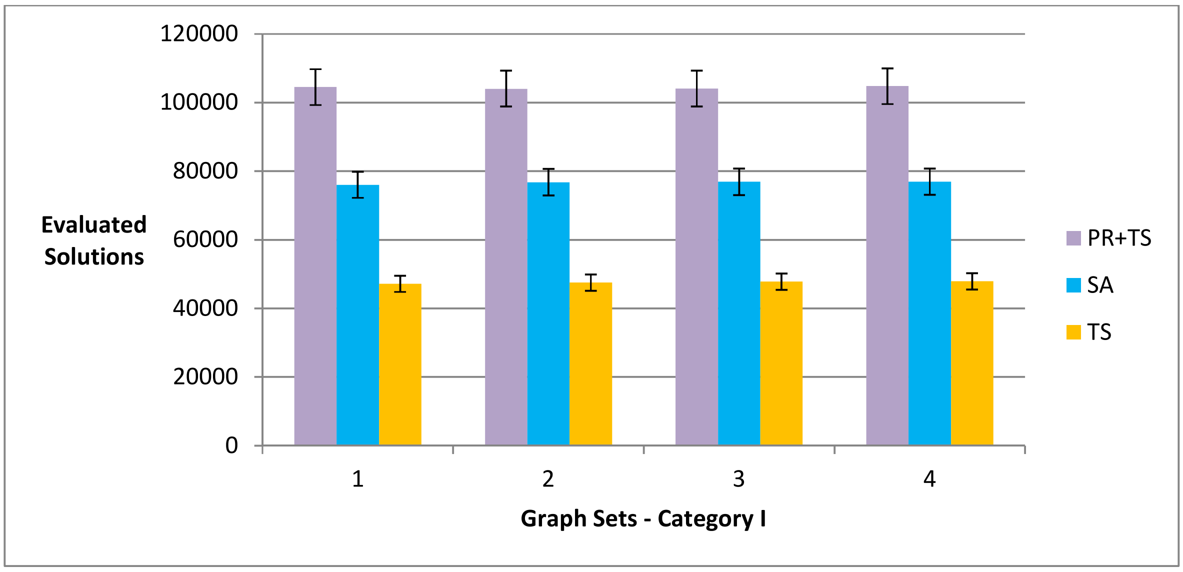


Figure 7.3 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of the 1st category (phase I)

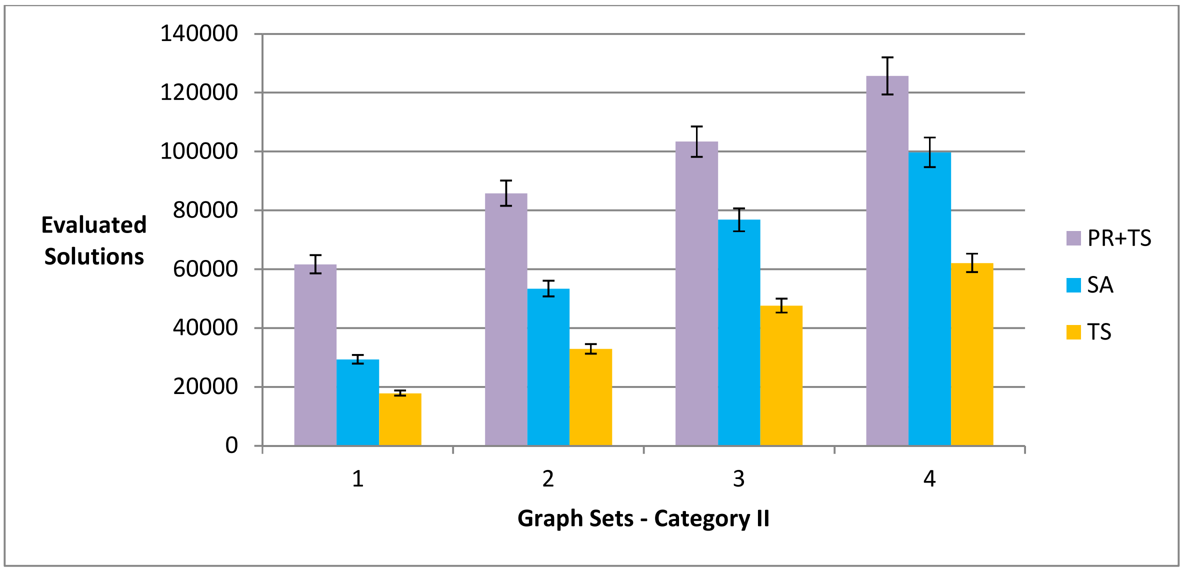


Figure 7.4 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of the 2nd category (phase I)

Figure 7.5 and Figure 7.6 demonstrate the execution time (in seconds) when the three methods were applied on the data of the first and second categories respectively. We conclude from these two figures that the execution time increases as the number of nodes in the graph increases as shown in the results of the data of the second category (Figure 7.6), unlike the data of the first category where the number of nodes is fixed.

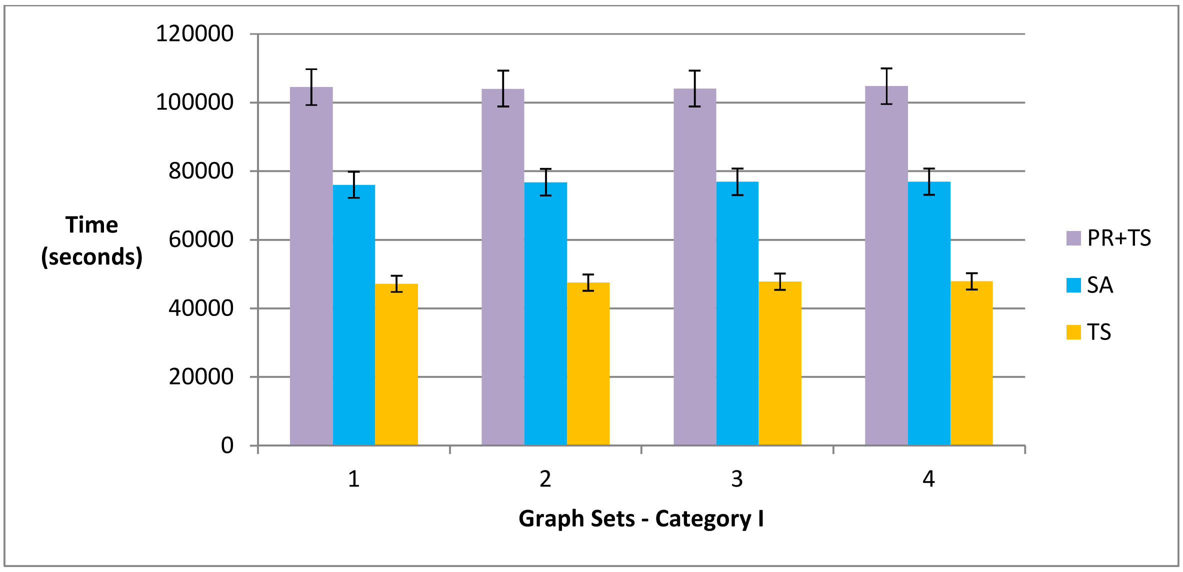


Figure 7.5 Bar chart with 95% confidence interval of execution time (in seconds) obtained by TS, SA, PR+TS when applied on the graphs of the 1st category (phase I)

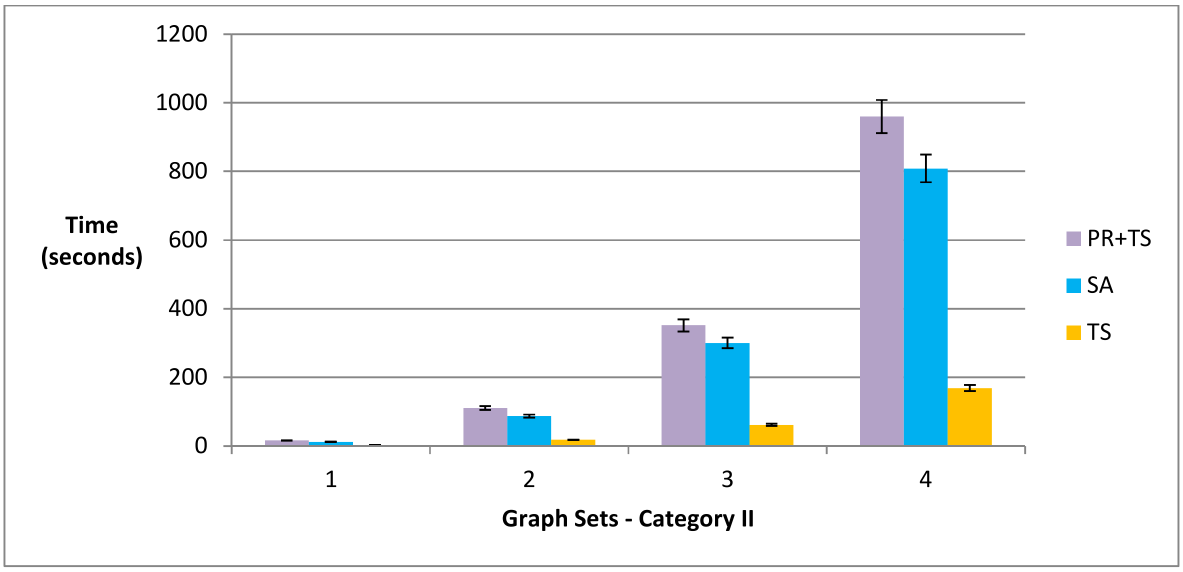


Figure 7.6 Bar chart with 95% confidence interval of execution time (in seconds) obtained by TS, SA, PR+TS when applied on the graphs of the 2nd category (phase I)

In order to analyse the overall performance of the three methods and to examine how good a layout can the methods achieve, we combined the results of both categories into one bar chart as presented in Figure 7.7 which shows the difference between the three methods in terms of the lowest fitness that can be obtained by each method. Another bar chart is presented in Figure 7.8 that shows the number of evaluated solutions required to reach those lowest fitness values.

The advantage of coupling tabu search with path relinking is very clear in Figure 7.7 as this combination had produced graph layouts with low fitness values compared to those layouts produced by pure tabu search and simulated annealing. However, the number of evaluated solutions performed by the coupled methods is larger than the other two methods, as shown in Figure 7.8. The statistical analysis of the fitness values and the number of evaluated solutions presented in Table 7.3 and Table 7.4 second this conclusion.

Note that the large number of evaluated solutions for the coupled methods is justified since the analysis of this phase is based on the overall performance of the methods when they run until they finish using the best values of parameters which were selected in the tuning process. The coupling of tabu search and path relinking requires many iterations in order to get the lowest fitness value that can be obtained. But this combination can still produce good graph layouts with a lower number of evaluated solutions as will be shown in the following two phases. The figures in Appendix B (B.1 and B.2) are samples of the layouts drawn by the three algorithms when applied on the graph datasets described in Table 7.1 and Table 7.2 respectively.

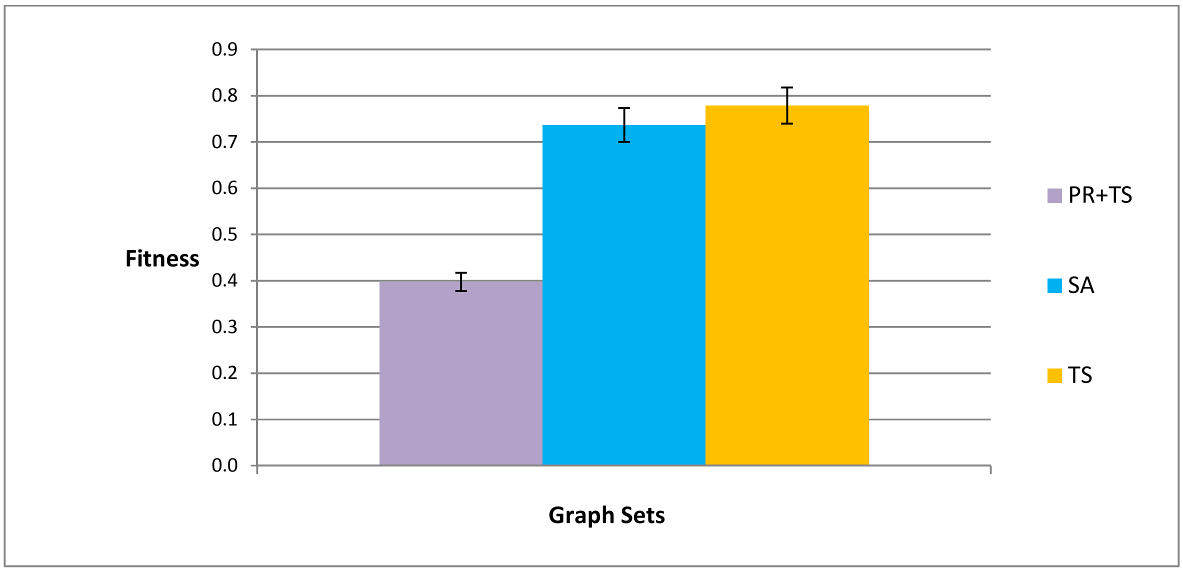


Figure 7.7 Bar chart with 95% confidence interval of the average overall fitness function obtained by TS, SA, PR+TS when applied on the graphs of both categories (phase I)

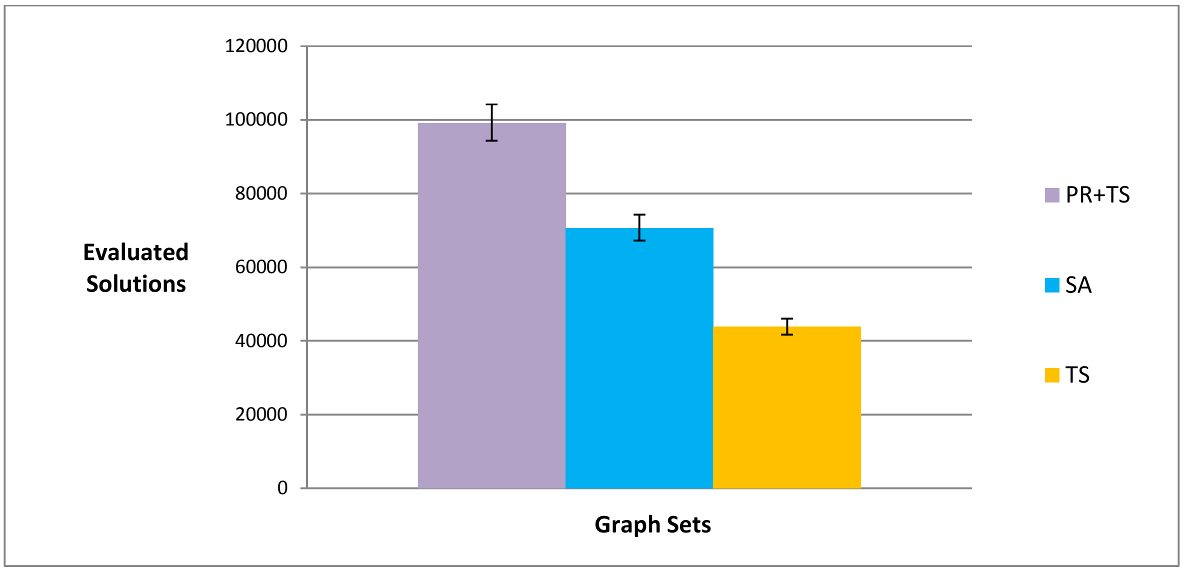


Figure 7.8 Bar chart with 95% confidence interval of the average overall number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of both categories (phase I)

Table 7.3 Statistical analysis of the fitness function for TS, SA, PR+TS when applied on the graphs of both categories (phase I)

	Fitness												
	PR + TS				**SA**				**TS**				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1C	0.242	0.242	0.272	0.216	0.493	0.492	0.509	0.486	0.506	0.505	0.549	0.463	1.38E-07
2C	0.382	0.382	0.403	0.359	0.772	0.772	0.782	0.756	0.825	0.821	0.923	0.741	5.33E-09
3C	0.468	0.469	0.525	0.408	0.905	0.907	0.913	0.886	0.951	0.956	0.988	0.906	2.06E-09
4C	0.585	0.593	0.629	0.538	0.999	1.000	1.006	0.991	1.042	1.038	1.084	0.988	5.33E-09
1D	0.315	0.314	0.360	0.266	0.353	0.355	0.368	0.338	0.398	0.388	0.591	0.328	8.74E-07
2D	0.300	0.294	0.333	0.276	0.598	0.596	0.613	0.583	0.634	0.628	0.700	0.592	5.33E-09
3D	0.397	0.399	0.437	0.362	0.792	0.791	0.803	0.786	0.857	0.846	1.148	0.782	1.25E-08
4D	0.489	0.487	0.536	0.431	0.984	0.991	0.999	0.938	1.021	1.028	1.080	0.946	5.06E-08
Overall	**0.397**	**0.398**	**0.437**	**0.357**	**0.737**	**0.738**	**0.749**	**0.721**	**0.779**	**0.776**	**0.883**	**0.718**	

Table 7.4 Statistical analysis of number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of both categories (phase I)

	Evaluated Solutions												
	PR + TS				SA				TS				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1C	104515	104487	108095	99665	76002	75983	76177	75874	47176	47181	47501	46872	2.06E-09
2C	104022	103990	109386	99175	76769	76755	76877	76635	47497	47499	47867	46776	2.06E-09
3C	104036	103916	110543	100263	76913	76924	77004	76757	47784	47786	48061	47555	2.06E-09
4C	104754	104428	111258	100026	76923	76910	77130	76804	47875	47936	48180	47550	2.06E-09
1D	61676	61705	62261	61077	29381	29390	29482	29278	17875	17902	18170	17572	2.06E-09
2D	85840	85667	91971	82419	53445	53459	53524	53352	32936	32880	33314	32680	2.06E-09
3D	103379	103465	107068	99410	76794	76801	76937	76681	47635	47674	48023	46603	2.06E-09
4D	125735	125386	131283	121829	99755	99763	100050	99565	62164	62118	62739	61676	2.06E-09
Overall	**99245**	**99130**	**103983**	**95483**	**70748**	**70748**	**70897**	**70618**	**43868**	**43872**	**44231**	**43410**	

7.2.2 Phase II

In this phase, we investigated the performance of the methods by counting the number of evaluated solutions performed by each method to reach similar values for the fitness function. Based on the results of the previous phase, we ran the tabu search first since it produced graph layouts with the largest fitness values compared to the other methods. This would easily allow the other methods to produce graph layouts with a quality which is at least as good as the quality of the ones produced by tabu search. Then, we ran the other methods until they reached an equal or better fitness value compared to the one reached by the tabu search. Finally, we measured the number of evaluated solutions for each method.

Figure 7.9 and Figure 7.10 present the number of evaluated solutions obtained when applying the three methods to reach graph layouts of a certain quality. Figure 7.11 and Table 7.5, on the other hand, show the average number of evaluated solutions obtained when the methods were applied on all the graph layouts of both categories along with the statistical analysis of the results.

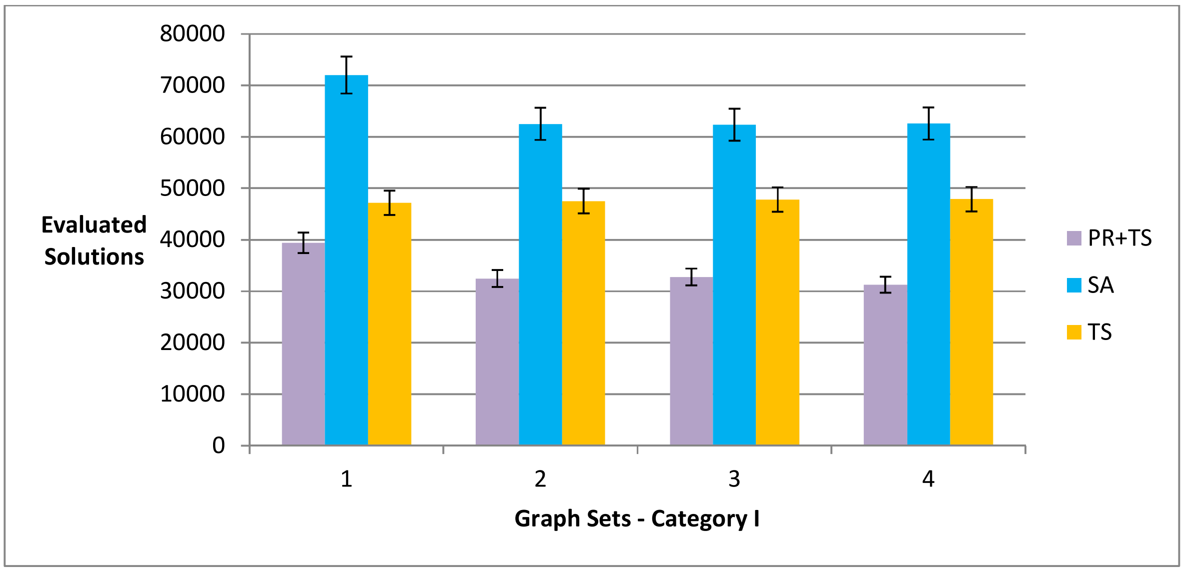


Figure 7.9 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of the 1st category (phase II)

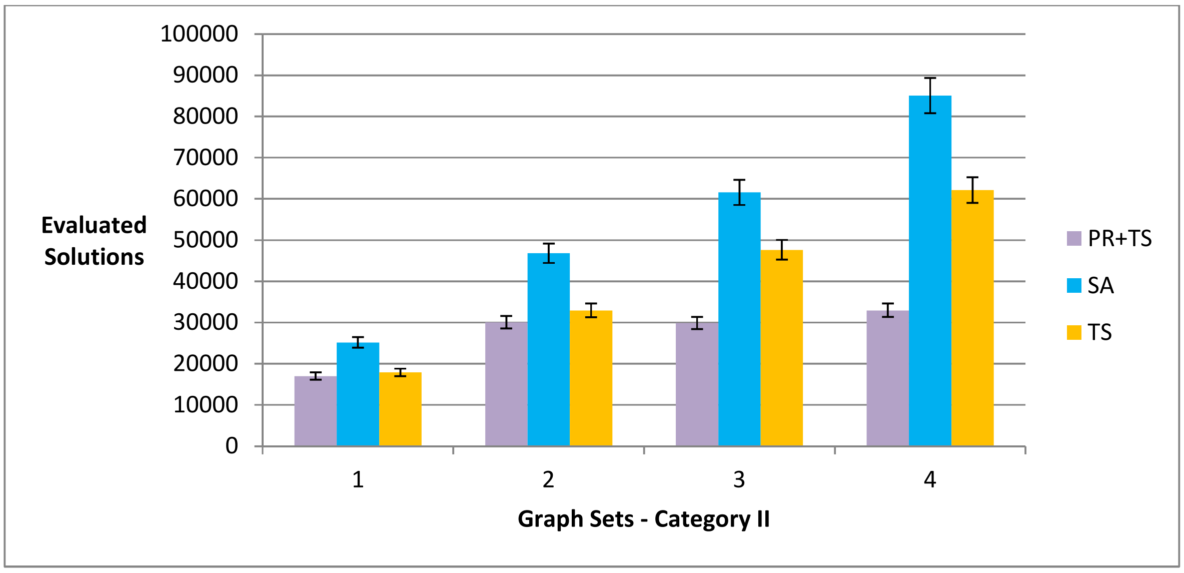


Figure 7.10 Bar chart with 95% confidence interval of the number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of the 2nd category (phase II)

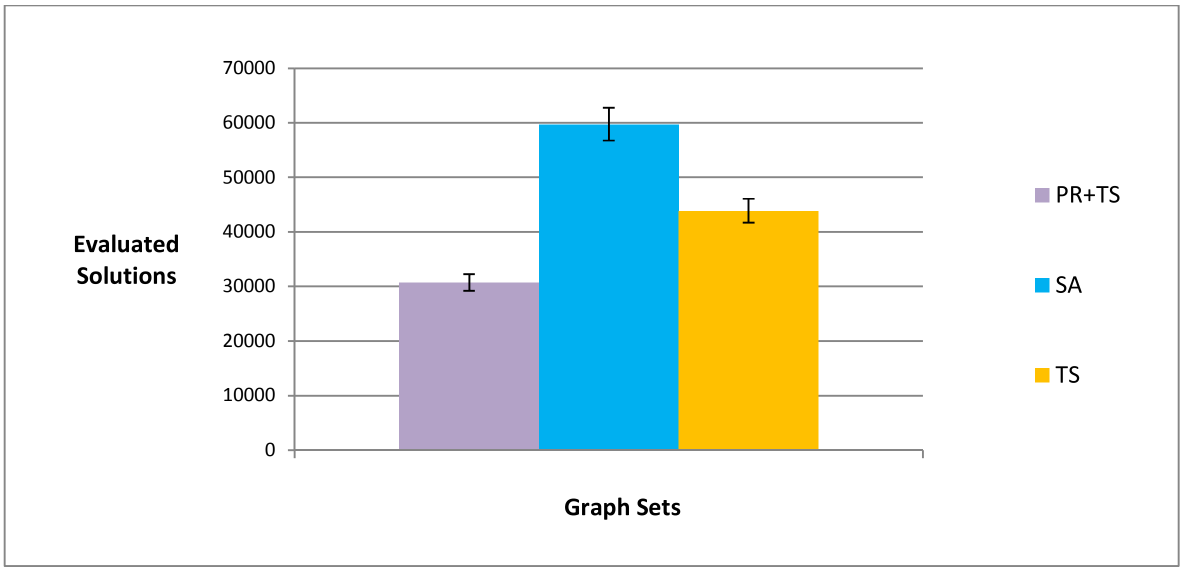


Figure 7.11 Bar chart with 95% confidence interval of the average overall number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of the two categories together (phase II)

Table 7.5 Statistical analysis of the average overall number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graphs of the two categories together (phase II)

	Evaluated Solutions												
	PR+TS				SA				TS				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1C	39423	36879	55180	29073	72012	73097	76190	65831	47177	47181	47501	46872	7.74e-06
2C	32470	30211	56041	14478	62492	62418	76734	48034	47497	47499	47867	46776	7.74e-06
3C	32777	29798	44884	21777	62331	60460	74949	54517	47785	47786	48061	47555	7.74e-06
4C	31268	29759	52479	23277	62579	62772	76923	53920	47875	47936	48180	47550	7.74e-06
1D	17021	17875	21032	8191	25147	26374	29433	13568	17876	17902	18170	17572	5.69e-05
2D	30058	30628	38544	17773	46816	48251	53497	35852	32936	32880	33314	32680	7.74e-06
3D	29876	29737	44593	8673	61580	61340	76733	37397	47636	47675	48023	46603	3.47e-04
4D	32981	29754	50216	21055	85040	83589	99629	68381	62165	62119	62739	61676	7.74e-06
Overall	**30734**	**29330**	**45371**	**18037**	**59750**	**59787**	**70511**	**47187**	**43868**	**43872**	**44232**	**43411**	**< 2.2e-16**

With reference to the bar charts in the figures and the results presented in the table, we conclude that coupling tabu search with path relinking could draw graph layouts with a certain quality by implementing a lower number of evaluated solutions with a significant difference to the number of solutions obtained by simulated annealing and pure tabu search. In other words, adding path relinking to pure tabu search improved the process of searching for good layouts with a lower number of evaluated solutions.

7.2.3 Phase III

In phase III, we investigated the quality of the layouts produced by the drawing algorithms. We tested which method produced graph layouts with the smallest fitness function values (best quality) when they perform the same number of evaluated solutions. We ran the tabu search method on the graphs for a predefined number of iterations (*maxIterations*) as described earlier in the parameters' tuning process presented in Chapter 4. We started with the tabu search because in phase I, it generated the lowest number of evaluated solutions. We ran the other methods until they perform the same number of evaluated solutions performed by the tabu search method. Finally, we measured the value of the fitness function produced by each drawing algorithm.

In Figure 7.12 and Figure 7.13, we show bar charts for the values of the fitness function when the three methods were applied to perform a set number of evaluated solutions on the graph layouts of the first and the second categories respectively. The average values of the fitness function obtained when we combined the results of applying the methods on both categories are presented in Figure 7.14 besides Table 7.6 which shows the statistical analysis of all the obtained results.

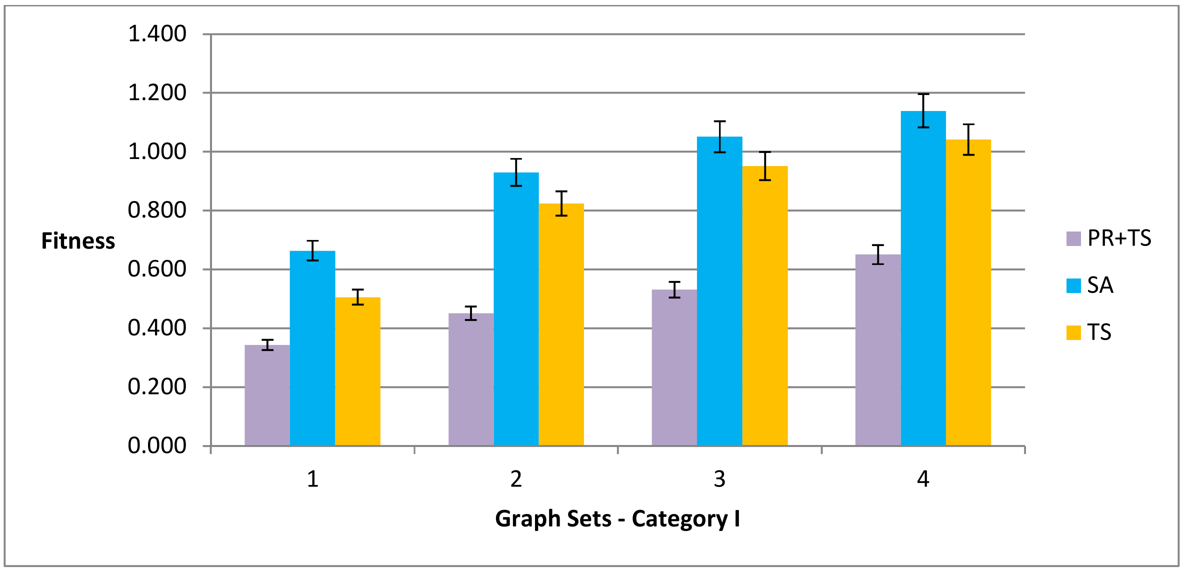


Figure 7.12 Bar chart with 95% confidence interval of the fitness function values obtained by TS, SA, PR+TS when applied on the graphs of the 1st category (phase III)

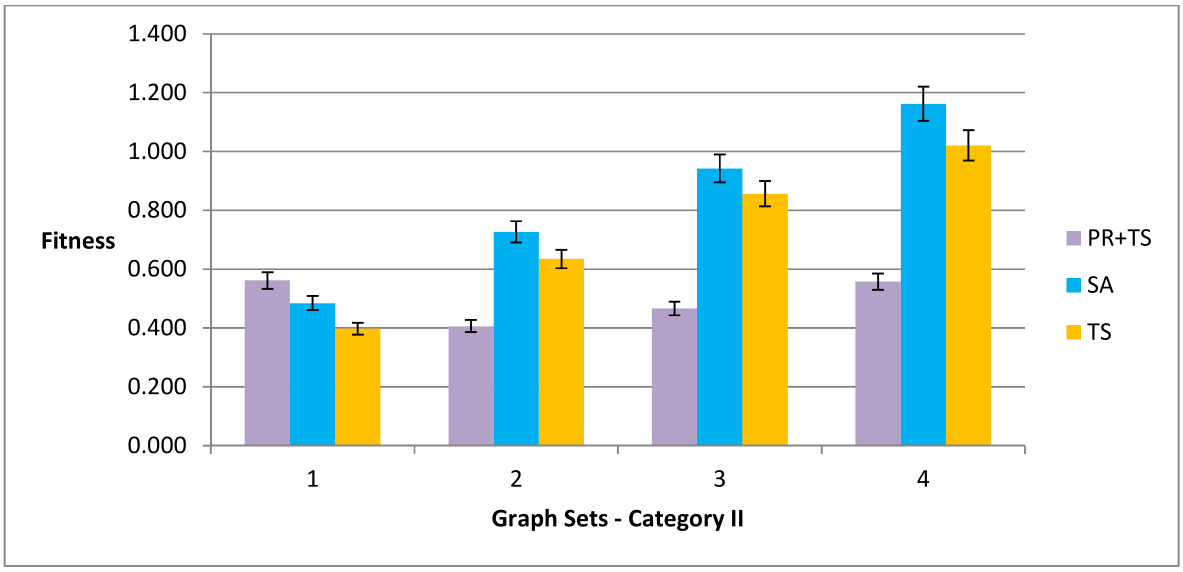


Figure 7.13 Bar chart with 95% confidence interval of the fitness function values obtained by TS, SA, PR+TS when applied on the graphs of the 2nd category (phase III)

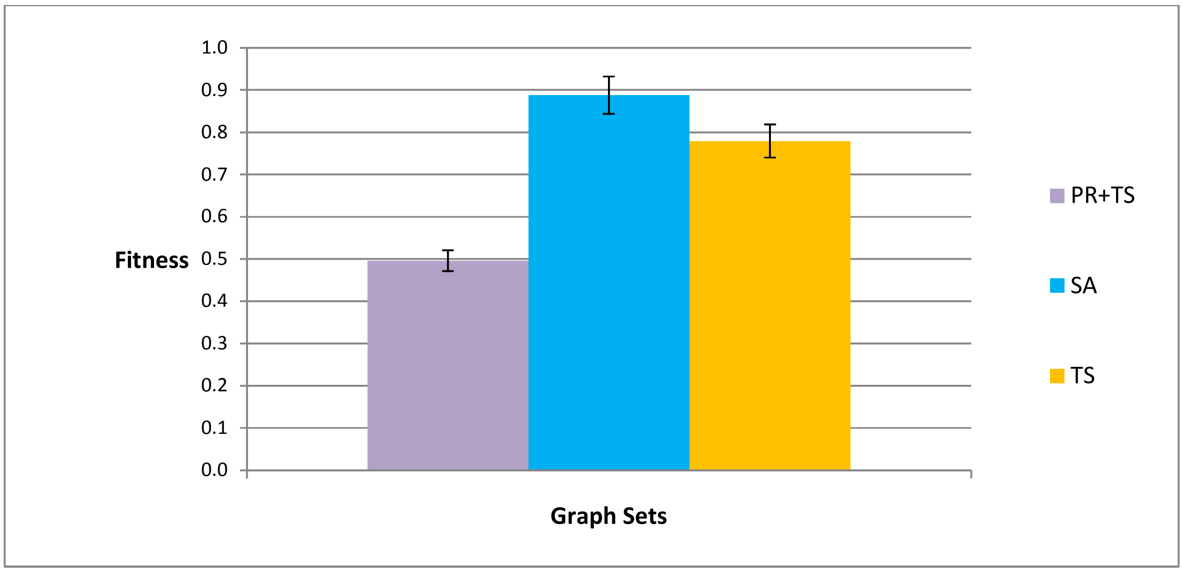


Figure 7.14 Bar chart with 95% confidence interval of the average overall fitness function values obtained by TS, SA, PR+TS when applied on the graphs of the two categories together (phase III)

Table 7.6 Statistical analysis of the average overall fitness function values obtained by TS, SA, PR+TS when applied on the graphs of the two categories together (phase III)

	Fitness												
	PR+TS				SA				TS				
Graph Set	Mean	Median	Max	Min	Mean	Median	Max	Min	Mean	Median	Max	Min	p-value
1C	0.343	0.347	0.384	0.320	0.664	0.663	0.676	0.646	0.506	0.505	0.549	0.463	7.74e-06
2C	0.451	0.451	0.481	0.419	0.930	0.930	0.945	0.912	0.825	0.821	0.923	0.741	7.74e-06
3C	0.531	0.532	0.595	0.474	1.051	1.052	1.064	1.038	0.951	0.956	0.988	0.906	7.74e-06
4C	0.650	0.656	0.692	0.595	1.139	1.139	1.152	1.117	1.042	1.038	1.084	0.988	7.74e-06
1D	0.561	0.564	0.631	0.440	0.485	0.486	0.507	0.464	0.398	0.388	0.591	0.328	5.69e-05
2D	0.407	0.404	0.458	0.377	0.727	0.729	0.741	0.709	0.634	0.628	0.700	0.592	7.74e-06
3D	0.466	0.469	0.510	0.425	0.942	0.946	0.959	0.920	0.857	0.846	1.148	0.782	5.69e-05
4D	0.558	0.557	0.612	0.493	1.162	1.167	1.176	1.123	1.021	1.028	1.080	0.946	7.74e-06
Overall	**0.496**	**0.497**	**0.545**	**0.443**	**0.887**	**0.889**	**0.902**	**0.866**	**0.779**	**0.776**	**0.883**	**0.718**	**< 2.2e-16**

Based on the results presented in the table and the previous three figures, we conclude that intensifying the search process of tabu search by introducing path relinking could lead to a quick investigation for graph layouts of good quality when compared with pure tabu search and simulated annealing performing the same number of evaluated solutions. This difference becomes significantly clear when the size of the graph increases as shown in Figure 7.13 that presents the results when the methods are applied on the graph layouts of the second category where the number of nodes in each set of graph layouts increases. However, the coupling of tabu search with path relinking does not seem to be very effective on small graphs when it is applied for a few number of iterations as shown in the first column of Figure 7.13.

Figure 7.15, Figure 7.16, and Figure 7.17 show three different examples of random graph layouts drawn by simulated annealing, tabu search, and tabu search coupled with path relinking.

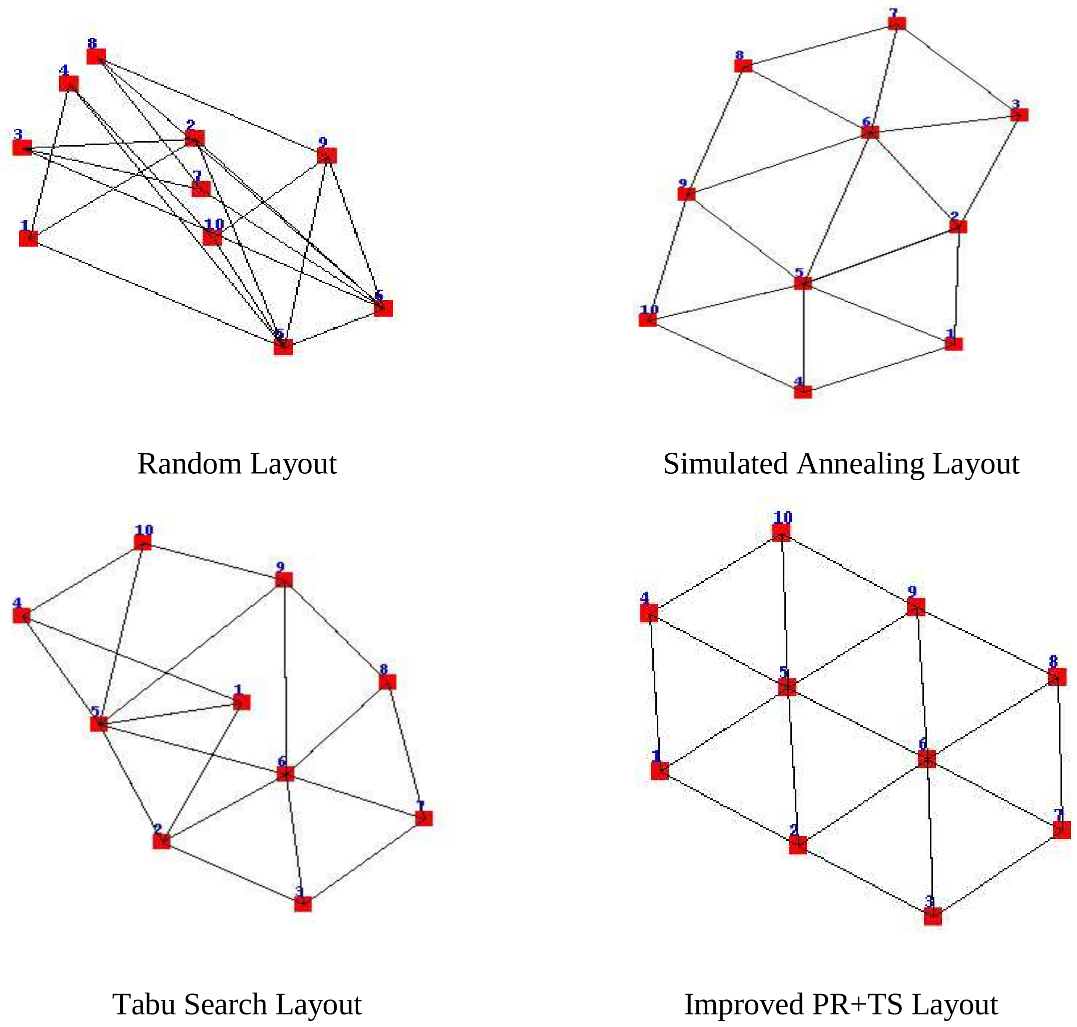
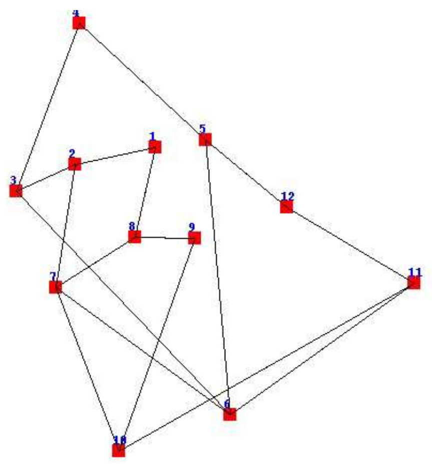
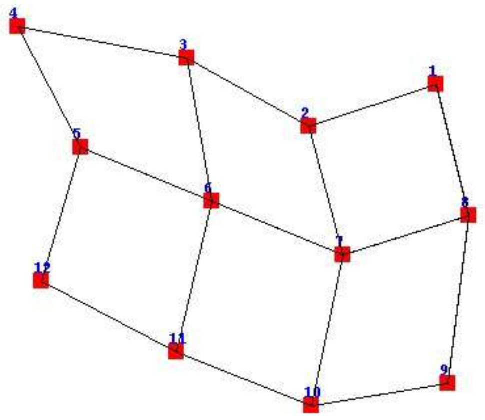
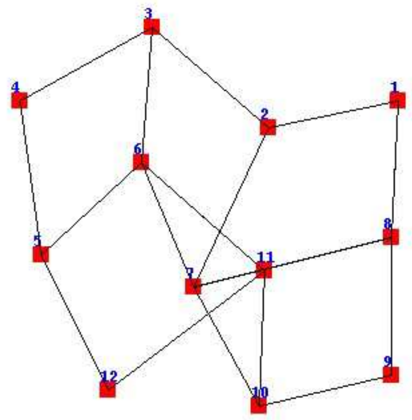


Figure 7.15 Example of connected graph layout with 10 nodes and 19 edges drawn within the canvas of our visualization tool by the three methods: SA, TS, PR+TS


Random Layout


Simulated Annealing Layout


Tabu Search Layout

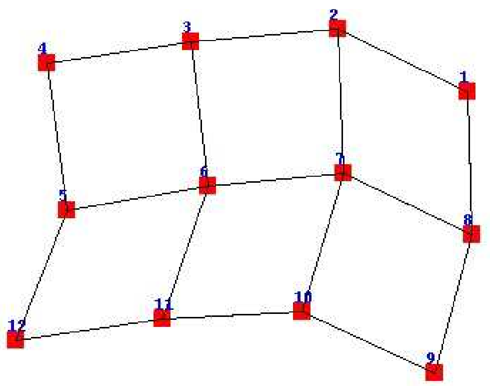

Improved PR+TS Layout

Figure 7.16 Example of connected graph layout with 12 nodes and 17 edges drawn within the canvas of our visualization tool by the three methods: SA, TS, PR+TS

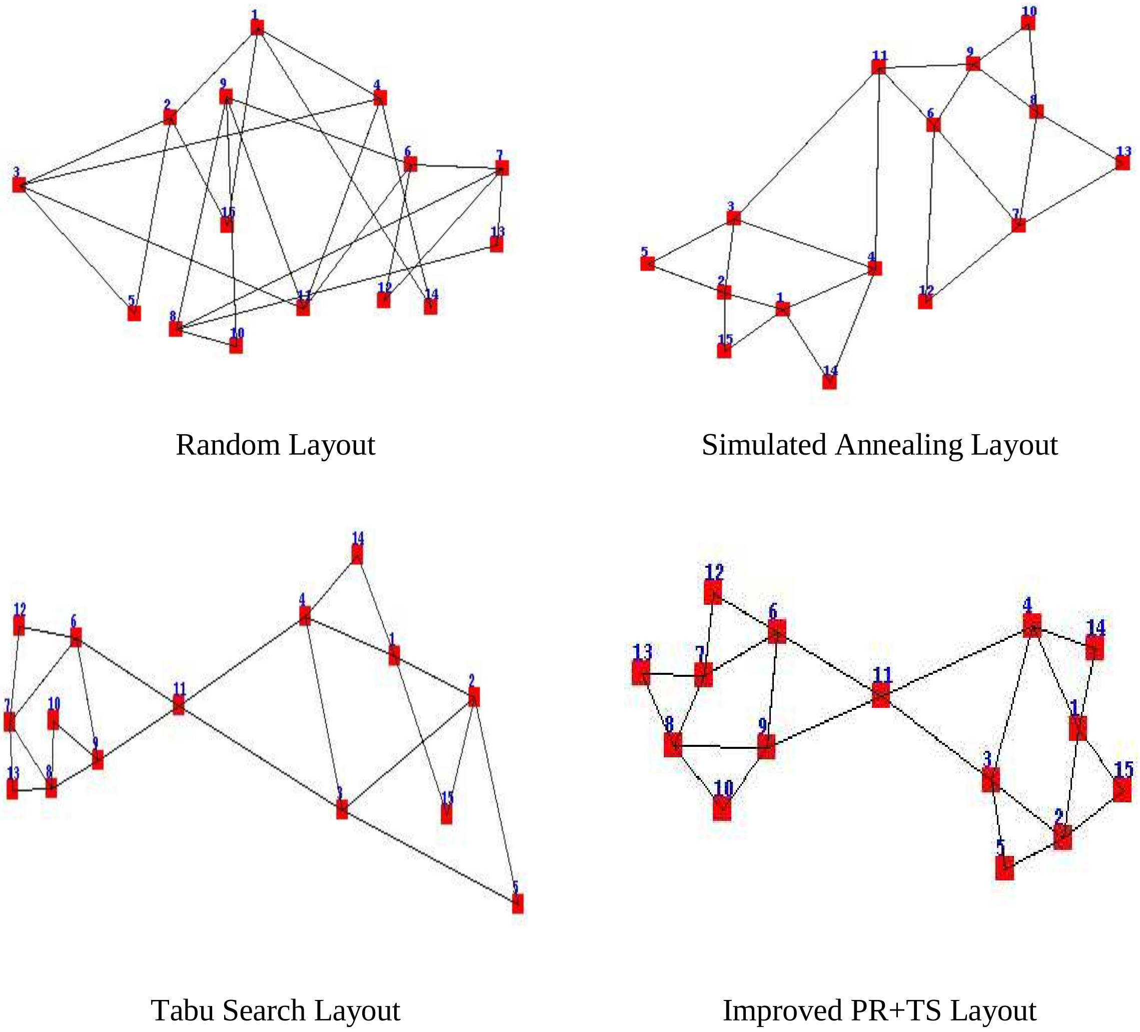


Figure 7.17 Example of connected graph layout with 15 nodes and 24 edges drawn within the canvas of our visualization tool by the three methods: SA, TS, PR+TS

7.2.4 Statistical Tests

In this section, we perform the same statistical tests which were described earlier in Chapter 5 (Section 5.2.4) to test the effect of randomness in generating the initial graph layouts used in comparing the methods. In order to show that there is a statistical significant difference in the results generated by the three methods, we applied the Friedman test since Shapiro-Wilk normality test showed that the population was not normally distributed. We ran the methods on 20 randomly generated test cases for each group of graphs in the first and the second categories. Simulated annealing, the only stochastic method, had run 30 times on each test case, and medians were calculated. Then we compared the three methods using the Friedman test with a significance level of 0.05. All the p-values shown in the last column of Table 7.3, Table 7.4, Table 7.5, and Table 7.6, are smaller than the value of our chosen significance level which concludes that there is a significant difference between the three methods.

With reference to the result of the Friedman test, pairwise comparisons between the three methods were performed using the Wilcoxon signed-rank test with Bonferroni correction, with a confidence level of 0.01, in order to show if there was a statistical significant difference between every pair of methods. Cohen's effect size measure was also applied. According to Cohen (1992), a small effect size is 0.2, a medium effect size is 0.5, and a large effect size is 0.8.

Table 7.7 and Table 7.8 show the p-values for the fitness function when applying the Bonferroni correction on the graphs of both categories according to the experiment conducted in phase I where each method would run without any restriction on the number of evaluated solutions or on the fitness value. The results indicate that the difference between each pair of methods is significant in terms of the quality of the generated layouts, except for special cases where the graph size is small as shown in the first graph dataset. Also, the effect sizes of fitness between path relinking coupled with tabu search against the other two methods are in the range of large and very large in most of the cases. On the other hand, there is a clear and a significant difference in the number of evaluated solutions between every pair of methods with very large effect sizes, as shown in Table 7.9 and Table 7.10.

Table 7.7 Effect size and p-values for the fitness function values after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 1st category (phase I)

		Fitness							
		Graph Set 1		**Graph Set 2**		**Graph Set 3**		**Graph Set 4**	
		PR + TS	**SA**	**PR + TS**	**SA**	**PR + TS**	**SA**	**PR + TS**	**SA**
SA	**p**	4.4e-11	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.8519	0	1.8725	0	1.8586	0	1.8641	0
TS	**p**	4.4e-11	0.2900	4.4e-11	1.2e-07	4.4e-11	1.2e-08	4.4e-11	1.2e-07
	effect	1.8422	0.5115	1.8161	0.9034	1.8516	1.3449	1.8724	1.2902

Table 7.8 Effect size and p-values for the fitness function values after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 2nd category (phase I)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	5.0e-06	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.1410	0	1.8675	0	1.8640	0	1.8674	0
TS	p	4.0e-07	0.0035	4.4e-11	2.3e-06	4.4e-11	1.5e-06	4.4e-11	0.0025
	effect	0.8384	0.5258	1.8214	0.9709	1.5666	0.5245	1.8429	0.7924

Table 7.9 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 1st category (phase I)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	2.0e-07	*	2.0e-07	*	2.0e-07	*	4.4e-11	*
	effect	1.8717	0	1.8302	0	1.7943	0	1.8117	0
TS	p	4.4e-11	2.0e-07	4.4e-11	2.0e-07	2.0e-07	2.0e-07	2.0e-07	2.0e-07
	effect	1.8765	1.8706	1.8563	1.8655	1.8419	1.8724	1.8502	1.8702

Table 7.10 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 2nd category (phase I)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	2.0e-07	*	4.4e-11	*	4.4e-11	*	2.0e-07	*
	effect	1.8704	0	1.8152	0	1.8556	0	1.8198	0
TS	p	2.0e-07	2.0e-07	4.4e-11	4.4e-11	4.4e-11	4.4e-11	4.4e-11	2.0e-07
	effect	1.8707	1.8705	1.8416	1.8721	1.8628	1.8599	1.8564	1.8697

When the Bonferroni test was applied on the results of phase II of the experiment, as shown in Table 7.11, Table 7.12, and Table 7.13, we see that path relinking outperformed simulated annealing in drawing graph layouts with similar objective function values using a limited number of evaluated solutions with very large effect sizes. It also outperformed the pure tabu search procedure on large graphs (as number of nodes increases) with very large effect sizes, unlike smaller graphs where there was no significant difference as shown in the first and the second graph datasets in Table 7.12.

Table 7.11 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 1st category (phase II)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	4.4e-11	*	5.2e-10	*	4.4e-11	*	4.4e-11	*
	effect	1.7531	0	1.4528	0	1.5998	0	1.6449	0
TS	p	0.0002	4.4e-11	8.7e-06	4.4e-11	4.4e-11	4.4e-11	3.6e-06	2.0e-07
	effect	0.9573	1.8459	1.1111	1.3504	1.4447	1.4673	1.5011	1.3799

Table 7.12 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 2nd category (phase II)

		Evaluated Solutions							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	3.4e-06	*	8.5e-09	*	8.7e-11	*	4.4e-11	*
	effect	1.1474	0	1.4358	0	1.4438	0	1.7678	0
TS	p	0.9800	3.4e-06	0.8700	4.4e-11	4.4e-11	7.2e-06	4.4e-11	4.4e-11
	effect	0.2286	1.3242	0.4833	1.6245	1.2565	1.1270	1.6876	1.5858

Table 7.13 Effect size and p-values for the number of evaluated solutions after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graph layouts of the two categories together (Phase II)

		Evaluated Solutions	
		PR+TS	SA
SA	p	4.4e-11	*
	effect	1.6120	0
TS	p	4.4e-11	4.4e-11
	effect	1.3000	1.5330

Table 7.14, Table 7.15, and Table 7.16 show the p-values for the values of the fitness function after conducting the Bonferroni test on the results of the experiment according to phase III where we ran the drawing algorithms so that they evaluate a specific number of solutions to test the quality of layouts that would be generated in a set time. The results in the tables show that coupling tabu search with path relinking draws graph layouts with better

quality compared to simulated annealing with very large effective sizes. It also outperforms pure tabu search as the size of the graph increases.

Table 7.14 Effect size and p-values for the fitness function values after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 1st category (phase III)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	4.4e-11	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	1.9000	0	1.8666	0	1.8619	0	1.8631	0
TS	p	4.4e-11	4.4e-11	4.4e-11	4.4e-11	4.4e-11	4.4e-11	4.4e-11	4.4e-11
	effect	1.8296	1.8142	1.7952	1.4303	1.8541	1.6960	1.8522	1.6389

Table 7.15 Effect size and p-values for the fitness function values after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graphs of the 2nd category (phase III)

		Fitness							
		Graph Set 1		Graph Set 2		Graph Set 3		Graph Set 4	
		PR + TS	SA	PR + TS	SA	PR + TS	SA	PR + TS	SA
SA	p	6.0e-06	*	4.4e-11	*	4.4e-11	*	4.4e-11	*
	effect	-1.2122	0	1.8830	0	1.8691	0	1.8671	0
TS	p	4.0e-08	1.2e-07	4.4e-11	4.4e-11	4.4e-11	9.9e-07	4.4e-11	4.4e-11
	effect	-1.4582	1.0635	1.7946	1.6529	1.5038	0.7855	1.8331	1.6902

Table 7.16 Effect size and p-values for the fitness function values after conducting the Bonferroni test on TS, SA, PR+TS when applied on the graph layouts of the two categories together (Phase III)

		Fitness	
		PR+TS	SA
SA	p	4.4e-11	*
	effect	1.9000	0
TS	p	4.4e-11	9.9e-07
	effect	1.8300	1.8140

Note that the improved method does not have any additional threats to validity more than those discussed earlier in Chapter 5 (Section 5.4). In the next section, we show how coupling tabu search with path relinking performs on real world graph datasets.

7.3 Experiments on Real World Graph Datasets

In this section, we want to show if the improved method can produce similar results in a real world setting by testing it against a standard public graph datasets described earlier in Chapter 5 (Section 5.3). We used the same 10 datasets listed in Table 5.15. The initial layout of the nodes in each graph was generated randomly. We tested the methods according to phases I, II, and III. The results of the experiments are shown in the following figures. Figure 7.18 and Figure 7.19 show the results of applying the methods on the real graph datasets according to phase I. The results shown in the charts second our conclusion in the previous section, that coupling path relinking with tabu search produces graph layouts with better fitness values compared to tabu search and simulated annealing. The difference becomes clearer as the size of the graph increases, as shown in Figure 7.18. However, that big difference requires more solutions to search in the neighbourhood and consequently, the number of evaluated solutions becomes larger, as shown in Figure 7.19.

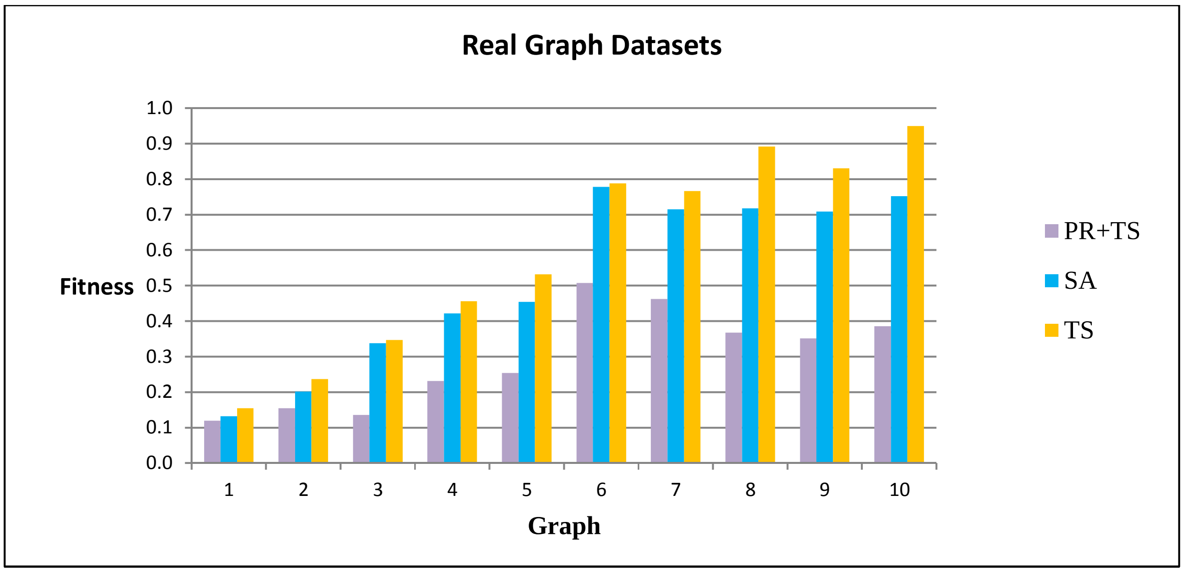


Figure 7.18 Bar chart of the fitness function values obtained by TS, SA, PR+TS when applied on the graph datasets in Table 5.15 (phase I)

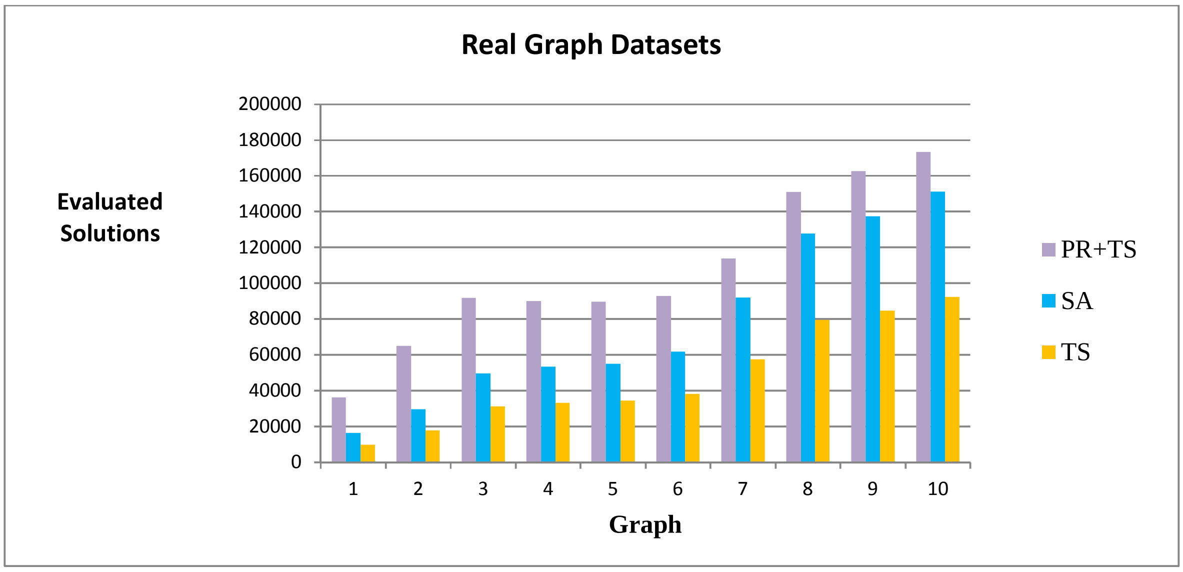


Figure 7.19 Bar chart of the number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graph datasets in Table 5.15 (phase I)

In Figure 7.20, where the experiment was based on phase II, we see that the improved method can reach the same fitness values of different graph layouts with a lower number of evaluated solutions compared to the other two methods. It also produces graph layouts with better fitness values, except for small graphs, when all the methods evaluate the same number of solutions as shown in Figure 7.21, where the experiment was based on phase III.

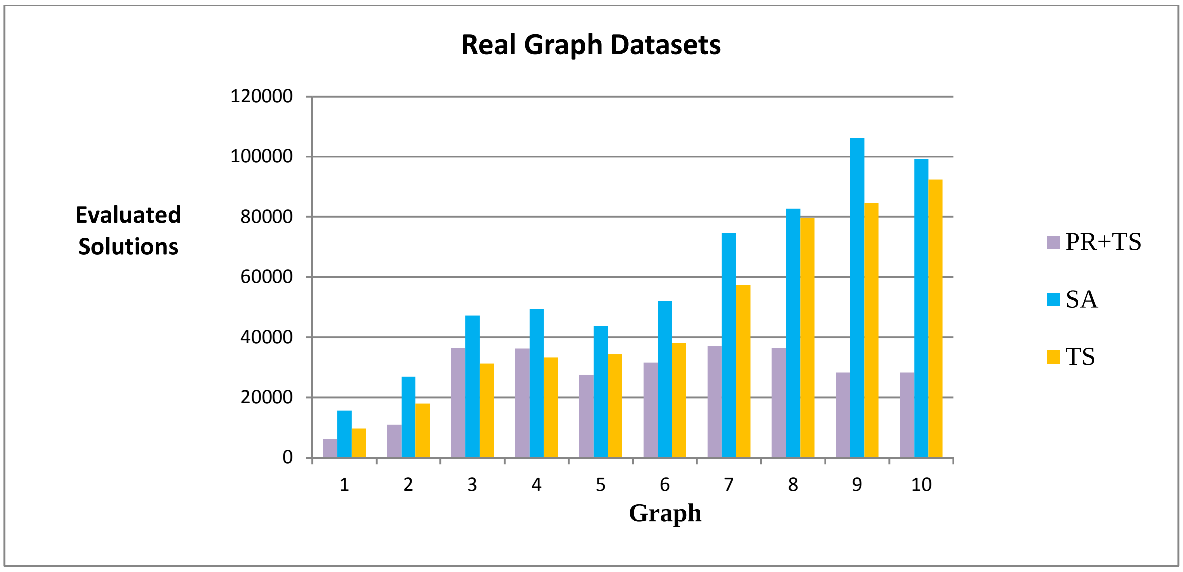


Figure 7.20 Bar chart of the number of evaluated solutions obtained by TS, SA, PR+TS when applied on the graph datasets in Table 5.15 (phase II)

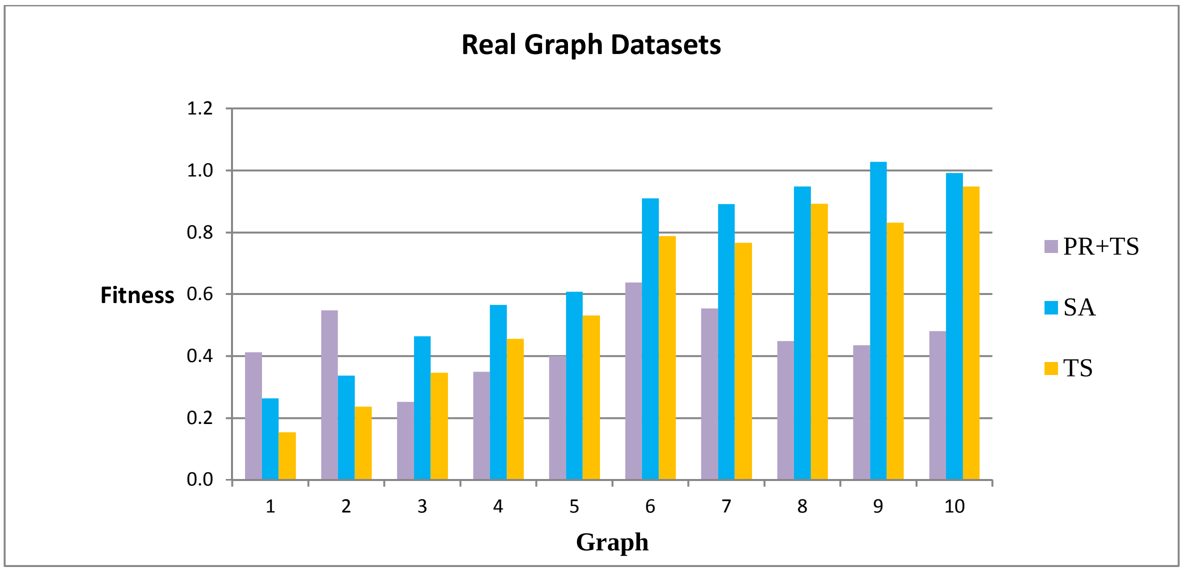
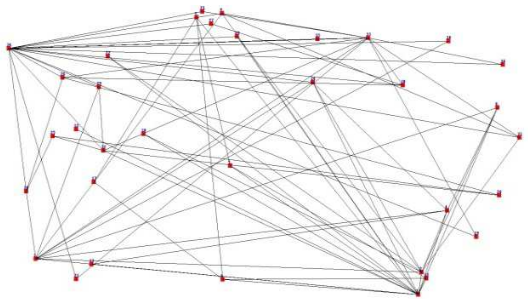
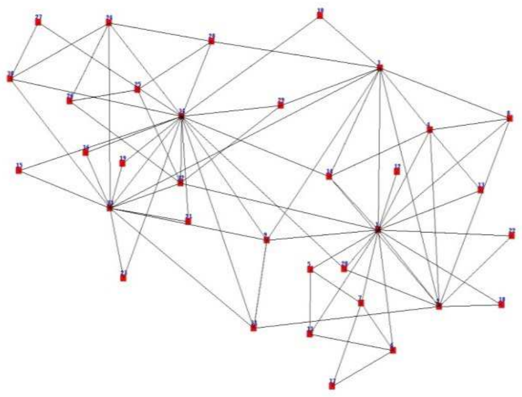
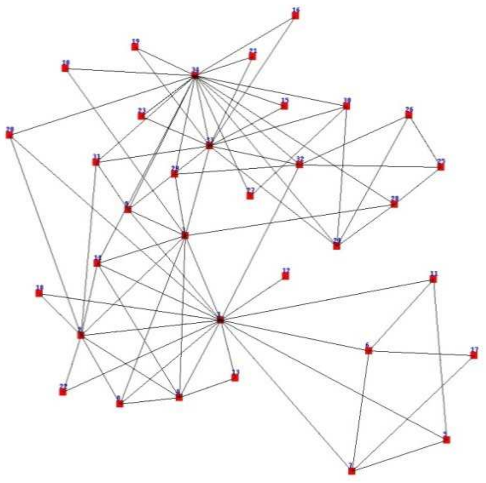


Figure 7.21 Bar chart of the fitness function values obtained by TS, SA, PR+TS when applied on the graph datasets in Table 5.15 (phase III)

Figure 7.22, Figure 7.23, Figure 7.24, and Figure 7.25 are four examples of the layouts produced by the methods when applied to graph datasets 1, 2, 3, and 5 respectively in the list of real world datasets described in Table 5.15.


Random Layout


Simulated Annealing Layout


Tabu Search Layout

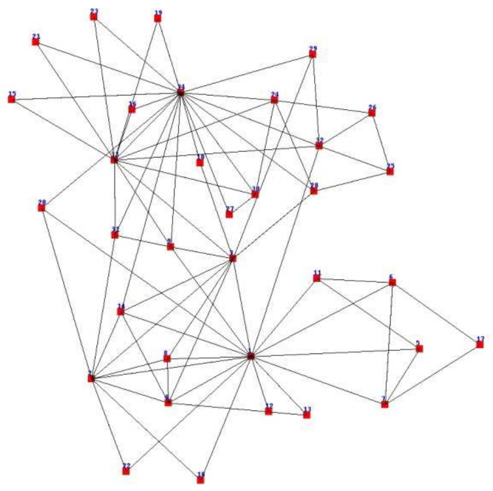

Improved PR+TS Layout

Figure 7.22 Layout of graph dataset 1 (listed in Table 5.15) produced by TS, SA, PR+TS drawn within the canvas of our visualization tool

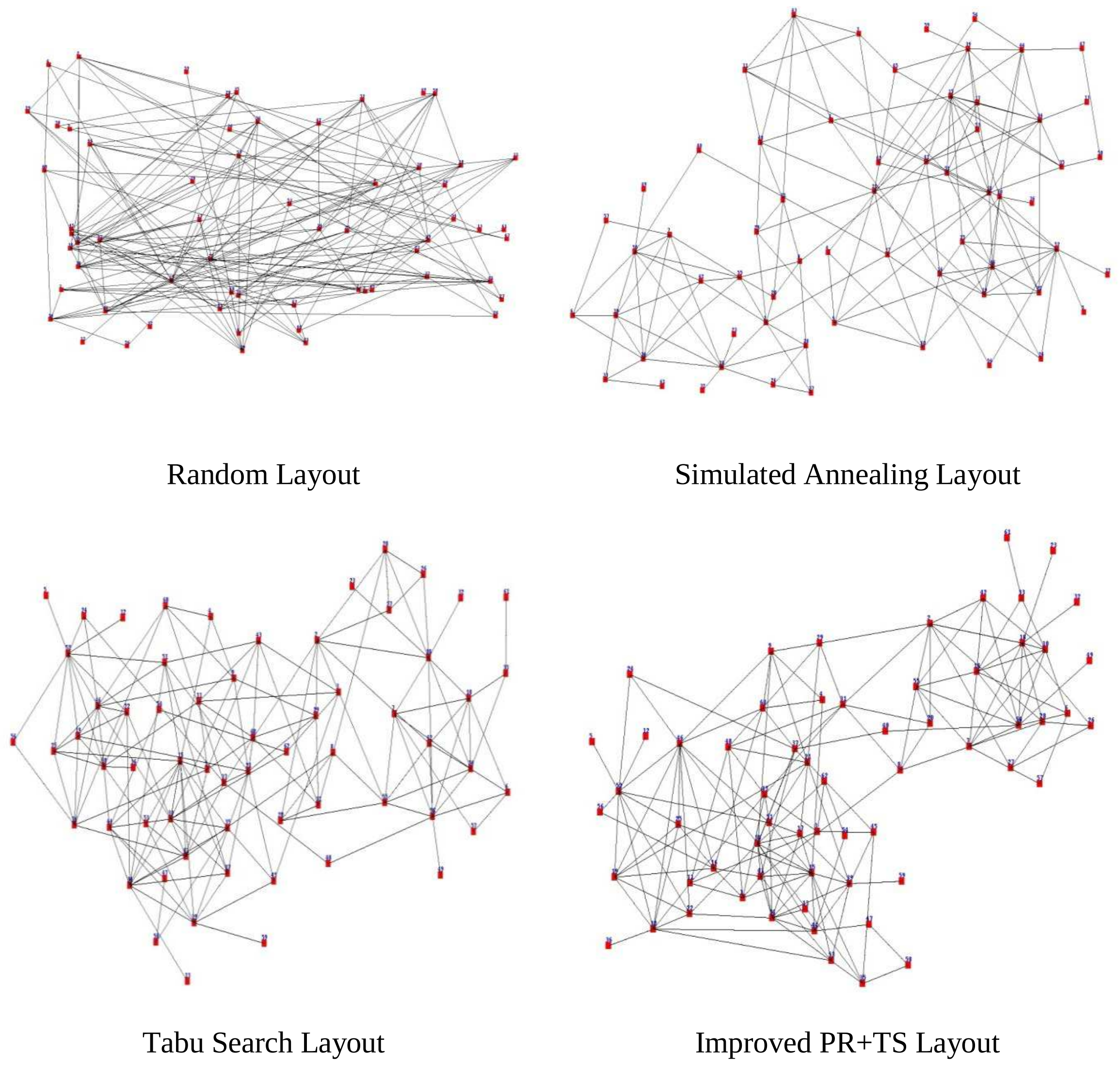
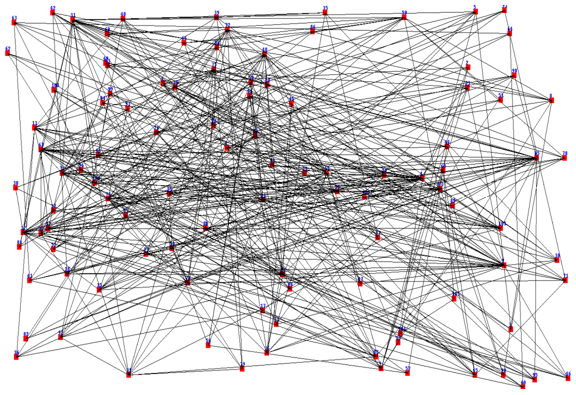
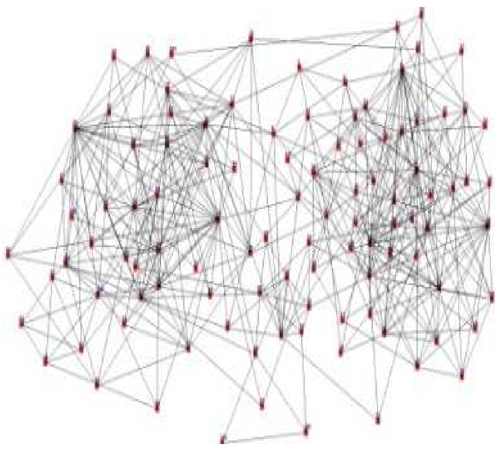
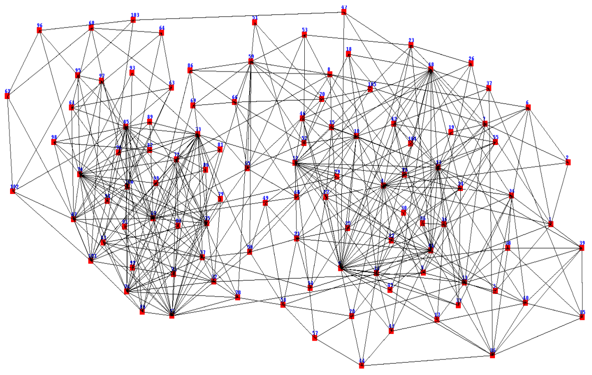


Figure 7.23 Layout of graph dataset 2 (listed in Table 5.15) produced by TS, SA, PR+TS drawn within the canvas of our visualization tool


Random Layout


Simulated Annealing Layout


Tabu Search Layout

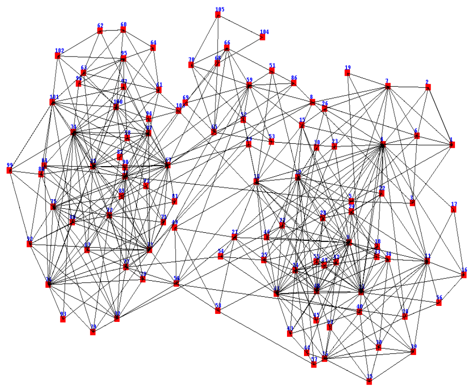

Improved PR+TS Layout

Figure 7.24 Layout of graph dataset 3 (listed in Table 5.15) produced by TS, SA, PR+TS drawn within the canvas of our visualization tool

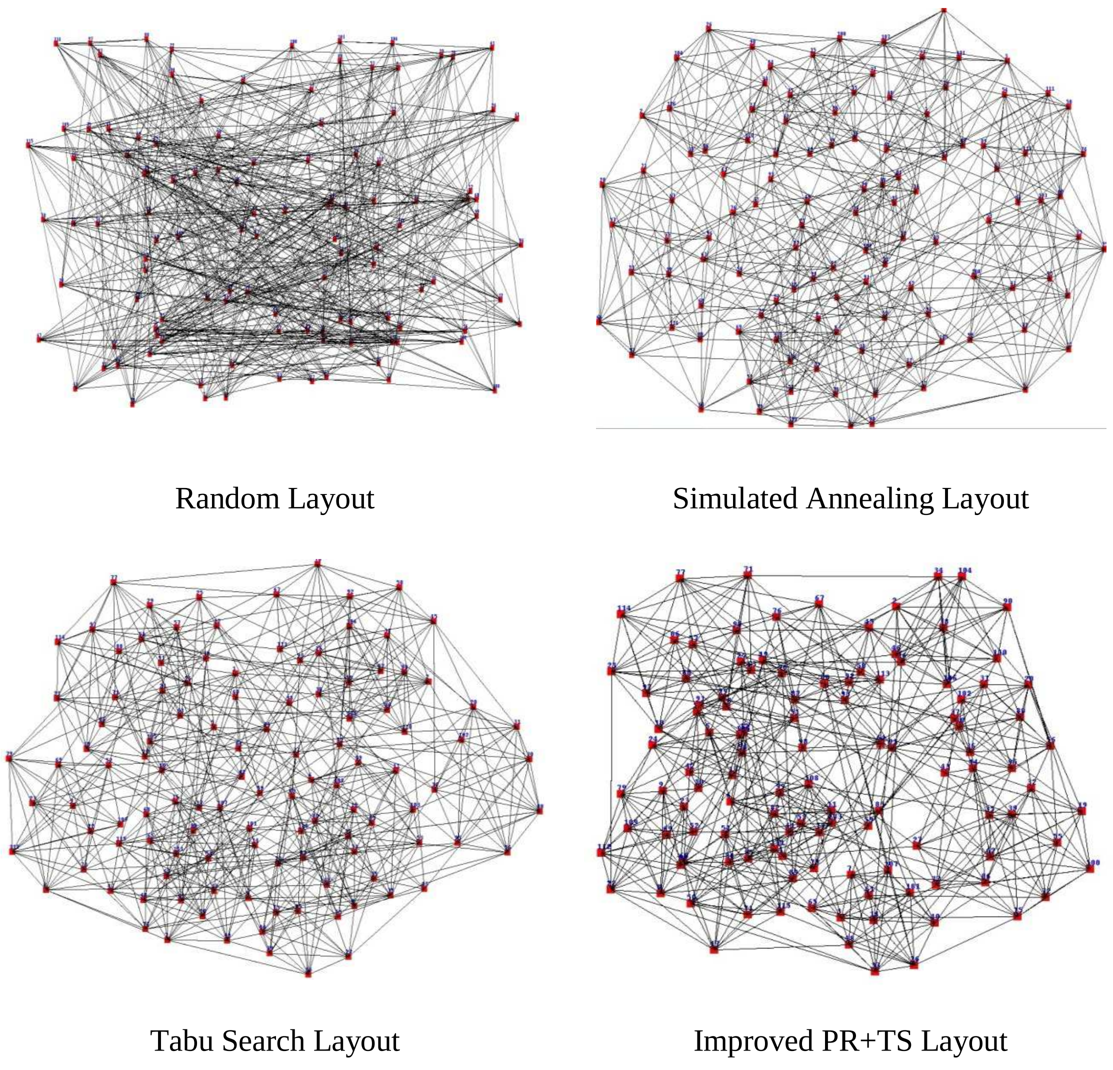


Figure 7.25 Layout of graph dataset 5 (listed in Table 5.15) produced by TS, SA, PR+TS drawn within the canvas of our visualization tool

In the next section, we analyse the performance of our method against the graph size accompanied with figures which describe its scalability. We also show its effect on each aesthetic criterion.

7.4 Scalability and Performance Analysis

In order to test the scalability of our method and its ability to work effectively on large graph datasets, we ran our method against simulated annealing on randomly generated large graphs, based on Erdos-Renyi model, according to phase I. Note that we excluded hill climbing from this comparison as the statistical tests in Chapter 5 showed that hill climbing is considerably worse than the other methods. We ran simulated annealing 30 times on each dataset, and the median value was recorded for each set. The graphs were generated using the same generator described in Chapter 3 and Chapter 5 (Section 5.1). We started with a graph dataset of 1000 nodes and 3003 edges and we kept increasing the number of nodes and edges as we move from one dataset to another as shown in Table 7.17. We stopped increasing the size of the datasets when we had a very long execution time for one of the tested methods (almost half a day).

Table 7.17 Characteristics of the graph datasets used in scalability testing

Graph Set	Nodes	Edges
1	1000	3003
2	1500	4503
3	2000	6003
4	2500	7503
5	3000	9003
6	3500	10503
7	4000	12003
8	4500	13503
9	5000	15002
10	5500	16503

Figure 7.26 shows that our method effectively minimises the value of the fitness function and outperforms simulated annealing regardless of how large the size of the graph is. Also, as Figure 7.27 and Figure 7.28 show, the speed of this minimisation process is efficient in our method compared to simulated annealing as the graph size increases. The figures show that increasing the number of nodes and edges (i.e. increasing the size of the graph) would increase the number of evaluated solutions and execution time for simulated annealing and our method as well, but with different rates of increase. Note that the execution time would be shorter if we test the methods for drawing graph layouts with a single criterion. However, since our fitness function contains multiple measures, it took a longer time to execute as some measures have a long computation time.

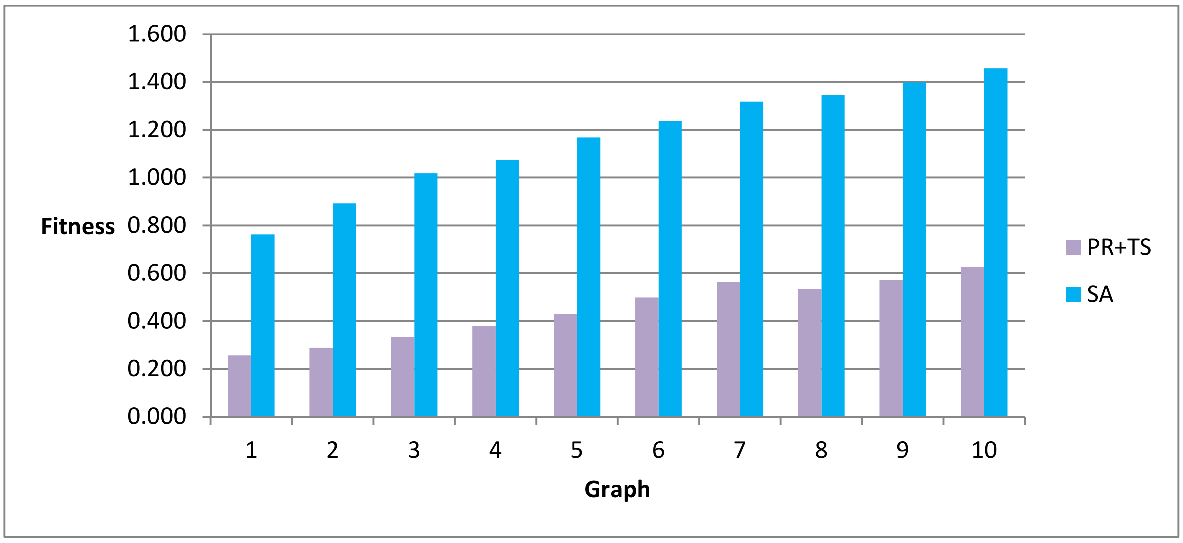


Figure 7.26 Bar chart of the fitness values obtained by PR+TS and SA when applied on graph datasets in Table 7.17 (phase I) for scalability testing

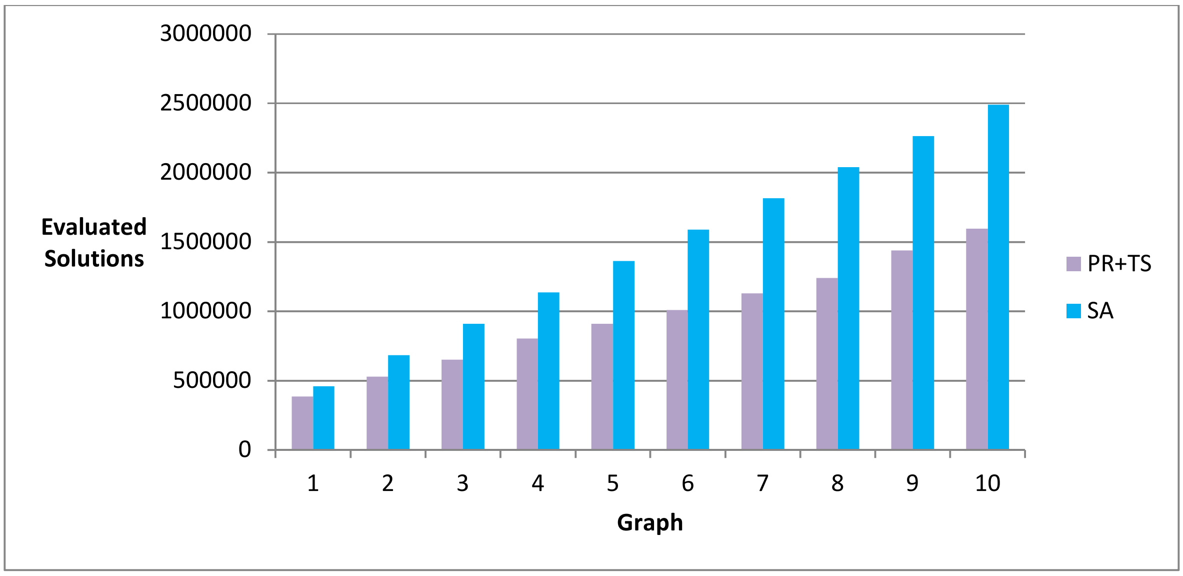


Figure 7.27 Bar chart of the number of evaluated solutions obtained by PR+TS and SA when applied on graph datasets in Table 7.17 (phase I) for scalability testing

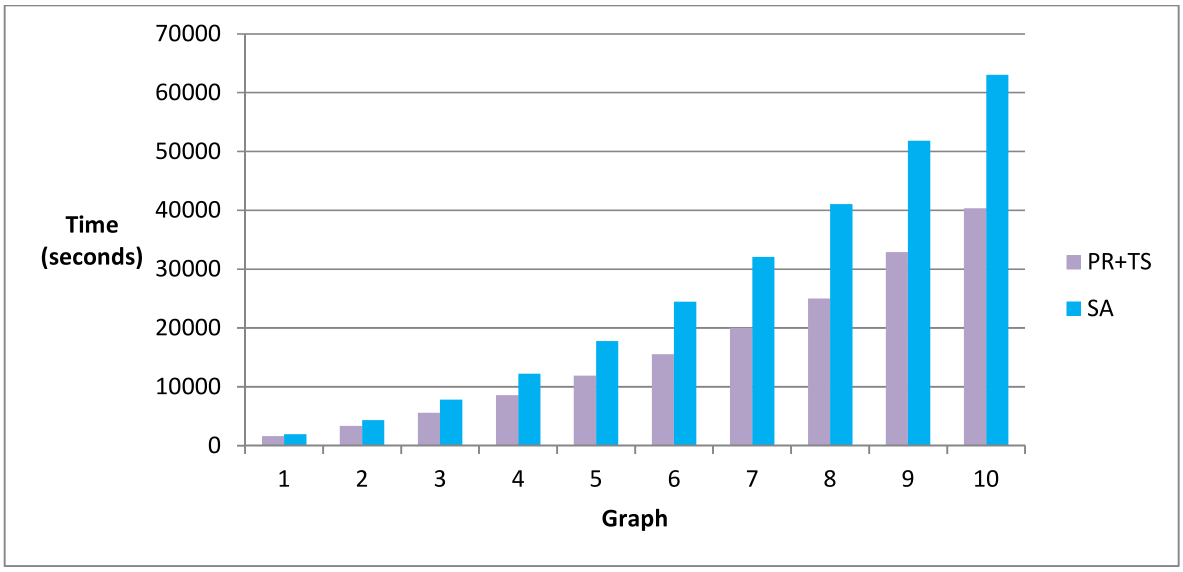


Figure 7.28 Bar chart of execution time in seconds obtained by PR+TS and SA when applied on graph datasets in Table 7.17 (phase I) for scalability testing

Figure 7.29, Figure 7.30, and Figure 7.31, show boxplots for the overall performance of our method when being applied on a set of graphs with an increasing number of nodes and edges, as described in Table 7.17, in terms of fitness values, number of evaluated solutions, and execution time in seconds respectively. All figures show that our method outperforms simulated annealing in all aspects.

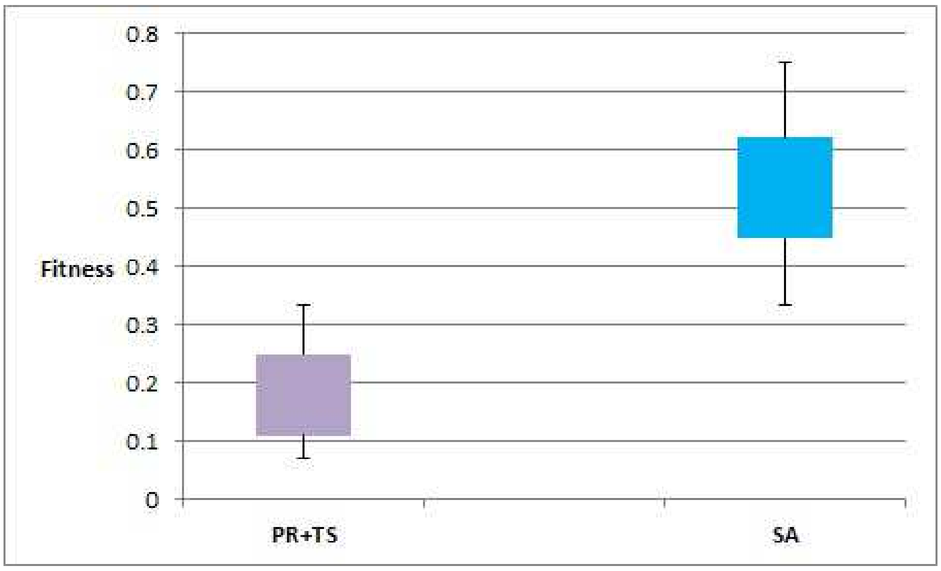


Figure 7.29 Box plot chart of the overall fitness values obtained by PR+TS and SA when applied on graph datasets with an increasing number of nodes and edges (Table 7.17)

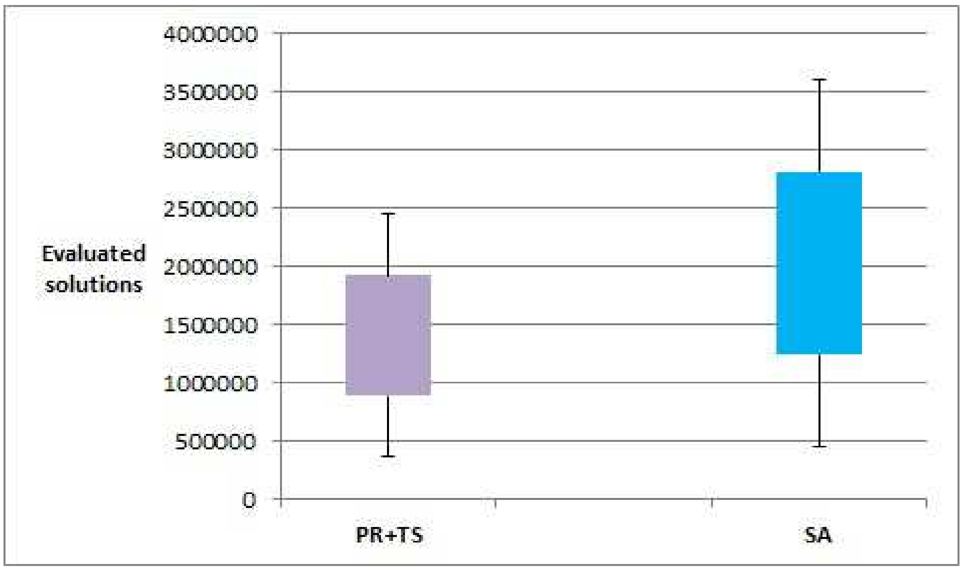


Figure 7.30 Box plot chart of the overall number of evaluated solutions obtained by PR+TS and SA when applied on graph datasets with an increasing number of nodes and edges (Table 7.17)

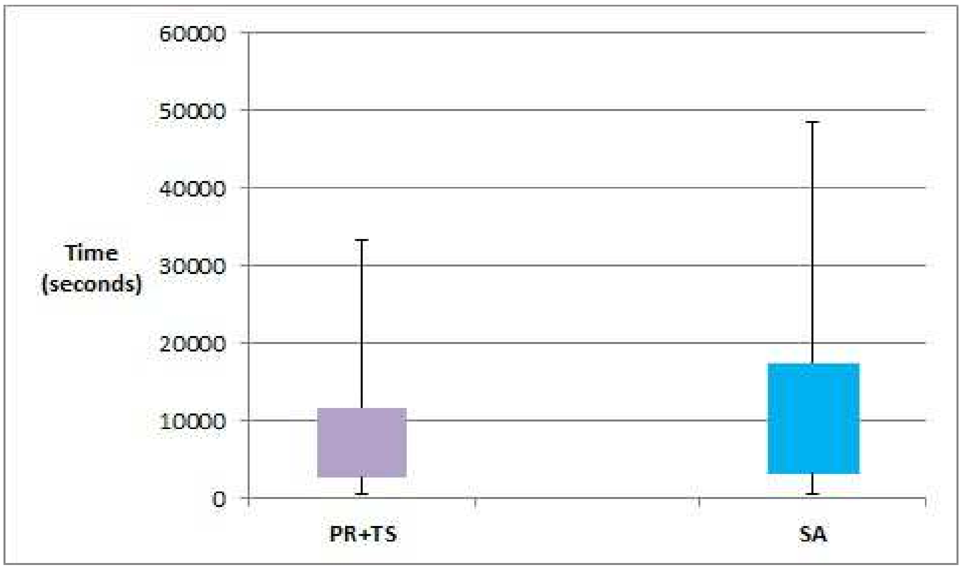


Figure 7.31 Box plot chart of the overall time in seconds obtained by PR+TS and SA when applied on graph datasets with an increasing number of nodes and edges (Table 7.17)

In order to examine the behaviour of our method on the value of the fitness function as the number of evaluated solutions increases, we ran the method on several graphs of the same size having 105 nodes and 441 edges but with different initial layouts. The average value of the fitness was recorded at different points during the execution time of the method. Figure 7.32 describes the algorithm’s behaviour by showing the change in the value of the fitness as the number of evaluated solutions increases. The figure shows that the fitness value decreases as the number of evaluated solutions increases.

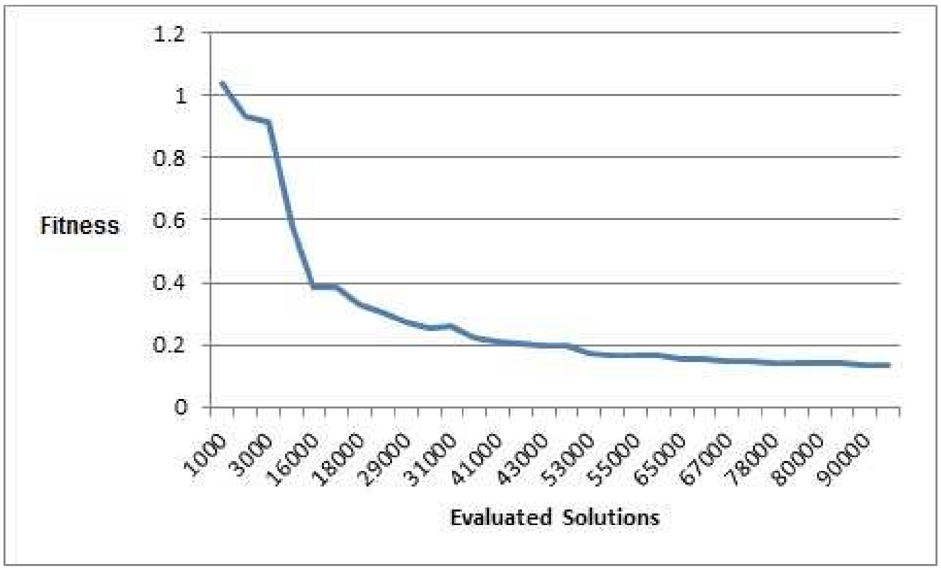


Figure 7.32 The change of the fitness value as the number of evaluated solutions increases

Last but not least, we find it interesting to report examples of the normalised values of each aesthetic used in our fitness function independently when being evaluated by hill climbing, simulated annealing, tabu search, and path relinking coupled with tabu search. Table 7.18 and Table 7.19 provide values from the real world datasets 3 and 5 described in Table 5.15.

Table 7.18 Normalised values of each aesthetic when the methods were applied on graph dataset 3 (listed in Table 5.15)

	node-node occlusion	edge length	edge crossings	angular resolution
HC	0.029602	0.119400	0.075273	0.249458
SA	0.021657	0.084227	0.038497	0.230175
TS	0.026453	0.061855	0.038879	0.219136
PR+TS	0.000279	0.024855	0.024902	0.085991

Table 7.19 Normalised values of each aesthetic when the methods were applied on graph dataset 5 (listed in Table 5.15)

	node-node occlusion	edge length	edge crossings	angular resolution
HC	0.079880	0.164624	0.058610	0.395361
SA	0.035921	0.077498	0.033709	0.312177
TS	0.048987	0.096359	0.039349	0.346663
PR+TS	0.000156	0.031453	0.026176	0.195369

7.5 Summary

In this chapter, we studied the effect of coupling path relinking with tabu search on the efficiency and the effectiveness of the proposed drawing algorithm. This was achieved by conducting three comparisons with the pure tabu search and simulated annealing drawing algorithms based on the quality of the layout that can be achieved by each drawing algorithm; the number of evaluated solutions performed by each method to reach a particular level of layout quality; and the quality of layout drawn by the methods after a fixed number of evaluated solutions. The experiments were conducted on new randomly generated datasets, different than those used in Chapter 5 in order to avoid overfitting, and on the same real world datasets. The statistical tests of the experiments on random graph datasets gave strong evidence that coupling path relinking with tabu search outperforms all the neighbourhood search methods discussed in this research in all aspects with very large effect size. The results of applying the methods on real world datasets support our conclusion. We also described the performance of the proposed method as the size of the graph increases and we showed that the method has better scalability compared to simulated annealing.

Chapter 8 Conclusions

This chapter provides a summary of the objectives and contributions. We also highlight a number of ideas which can be explored in the future.

8.1 Objectives and Contributions

In this work, we addressed the research area of graph drawing, where the main task was improving the efficiency and effectiveness of neighbourhood search-based methods for drawing general graph layouts with undirected straight lines based on a weighted sum multi-criteria fitness function. This approach has the advantage of allowing explicit combinations of metrics that can be tuned to meet user preferences. We described a novel automated neighbourhood search method based on tabu search and path relinking that have not been used before in drawing general graph layouts with multi-aesthetic criteria, unlike hill climbing and simulated annealing.

To achieve our goals, we started with implementing a visualisation tool that we used for testing all the neighbourhood search methods discussed in this thesis. The tool allowed the user to choose the preferred values of the parameters for each method, and the weights of each aesthetic metric (Chapter 3). It was not possible to determine unified weights that work well for all types of graphs, and indeed weights could vary according to application area and user preferences. Hence, we assigned the value 1 to all the weights for a fair comparison between the methods.

The first attempt for improving neighbourhood search methods in the field of drawing graph layouts was made by implementing an automated drawing method based on tabu search. The method searches for the best positions of the nodes, so minimising the value of the fitness function and drawing a nice graph layout. The key feature in tabu search was the combination of forbidding reverse moves using a memory-based tabu list and allowing escapes from local optima. We also implemented the basic neighbourhood search-based graph drawing algorithms for hill climbing and simulated annealing in order to be compared with our method. Besides the fact that all the methods shared the same local (neighbourhood) search space, we followed a unified systematic incremental procedure for tuning the values of the parameters of each method to select proper values which produce graph layouts with small fitness values (good quality). We provided figures and tables which describe the effect of adjusting the value of each parameter on the quality of the layouts and the efficiency of the drawing algorithms (Chapter 4).

We then conducted a comprehensive comparison between the three neighbourhood search-based drawing algorithms: hill climbing, simulated annealing, and tabu search. The comparison was broken down into three phases to answer the following questions: How good a layout can be achieved by each drawing algorithm? How many evaluated solutions performed by each method to reach a particular level of layout quality? And how good is the quality of layout drawn by the methods after a fixed number of evaluated solutions? We provided quantitative evidence of experimental results on randomly generated graph layouts, based on Erdos-Renyi model, and real world graphs to assert that the tabu search approach can draw a graph layout with a good quality in a smaller number of evaluated solutions compared to the hill climbing and the simulated annealing approaches. We also conducted statistical tests that showed, along with the large effect sizes, that the tabu search drawing algorithm was faster than the hill climbing drawing algorithm. It produced, along with simulated annealing, graph layouts with better quality regardless of the graph size in terms of number of nodes and edges. In addition, the efficiency of our tabu search-based method was better than the simulated annealing algorithm but the latter produced graph layouts with similar or slightly better fitness values compared to those produced by our tabu search algorithm when both methods ran without limitations on the number of evaluated solutions (Chapter 5).

Since our tabu search drawing algorithm had not outperformed simulated annealing in some aspects, we improved our method by coupling it with path relinking. The desire to tunnel through blocked off areas created by tabu search solutions was the main reason of choosing path relinking procedure to couple with tabu search to intensify the searching process between an initial and a guiding solutions selected from a set of elite solutions generated by the tabu search drawing algorithm. The integration of features of tabu search and path relinking in one implementation made our method a more effective graph layout method than the other neighbourhood search methods. Building a reference set of elite solutions generated by tabu search and moving efficiently along the path between two solutions were the main two aspects of our path relinking procedure. We also developed a systematic way for choosing the values of the parameters used by the method. We performed one round of parameter tuning to adjust the values of the parameters of the basic path relinking procedure. Then, we proposed two improvements on the basic implementation: a proper selection of the initial and the guiding solutions from the reference set of elite solutions; and an improved neighbourhood searching strategy based on a variable step size. The proposed improvements introduced two new parameters that could affect the performance of the procedure. Therefore, we performed a final round of parameters calibration process in order to assign reasonable values for each parameter before we examine the performance of our improved neighbourhood search method compared to other neighbourhood search methods (Chapter 6).

Finally, we studied the effect of coupling path relinking with tabu search on the efficiency and the effectiveness of the proposed drawing algorithm. This was achieved by conducting three comparisons, with our tabu search drawing algorithm and simulated annealing drawing algorithm. Our experimental results on random graphs and real world graphs showed that our tabu search/path relinking approach draws graph layouts with good quality in a relatively low number of evaluated solutions. Coupling tabu search with path relinking outperformed all the other methods discussed in this work in both terms of quality of layout and speed of layout process with very large effect sizes. We also described the performance of the proposed method as the size of the graph increases and we showed that the method had a better scalability when compared against simulated annealing (Chapter 7).

8.2 Future Work

In this section, we list a number of potential ideas which can be investigated to extend the work covered in this thesis.

1. Experiments can be conducted to study the efficiency of this method when applied to different types of graphs such as trees, hierarchical, and circular graphs. Our method can be easily adjusted to work with directed edges, but each type of these graphs has its own aesthetic measures such as: subtree separation, closest and farthest leaves for tree graphs; uniform edge direction and cycle removal for hierarchical graphs; partitioning the graph into clusters and placing the nodes of each cluster onto the perimeter of an embedding circle for circular graphs (Tamassia 2013). These aesthetics, in addition to the ones discussed in this thesis which usually exist in any

graph, must be formulated in a weighted sum multi-criteria objective function to be optimised by our proposed method.

2. The performance of our method can be further improved by implementing a hybrid of path relinking and a Greedy Randomized Adaptive Search Procedure (GRASP). This combination has been previously applied efficiently in some applications with promising results (Laguna & Marti 1999). In GRASP, each iteration consists of constructing a candidate solution and then improves that solution by applying an exchange procedure to find a local optimum. ‘The construction phase is iterative, greedy, randomized, and adaptive. It is iterative because the initial solution is built considering one element at a time. It is greedy because the addition of each element is guided by a greedy function. It is randomized because the selection of that element is made in a random fashion. And it is adaptive because the element chosen at any iteration in a construction is a function of those previously chosen. The improvement phase typically consists of a local search procedure (Duarte et al. 2017). Unlike tabu search, the generated solution by each GRASP iteration is not linked to the next solution by a sequence of neighbourhood moves. Therefore, the relinking process can have different interpretations with GRASP (Fleurent & Glover 1999).

3. There is a relationship between the algorithm’s execution time (in seconds) and the calculation of each metric in the fitness function (Davidson & Harel 1996). In our implementation, when the fitness function is evaluated, the aesthetic measure is recalculated for all nodes and edges. This slows down the execution time (but not the number of evaluated solutions). The runtime could be improved if we use memoisation on the calculation of metrics by storing previous values and calculating the metric only for the nodes and edges that are affected by a movement in the neighbouthood search space.

4. More investigations can be performed on the effectiveness of our approach in comparison with force-directed approaches and other population-based approaches that have been previously used in the field of graph drawing such as Genetic Algorithms (Eloranta & Mäkinen 2001; Vrajitoru 2009) and Ant Colony optimisation (Ware & Richards 2013).

5. An empirical study on human users could be conducted to evaluate the layouts generated by different graph drawing algorithms as visualisation is also concerned with how significant the differences are to the human eye and the human sense of aesthetics.

6. One way to improve the quality of solutions in tabu search is to divide the sets into Pareto and candidate lists (Baykasoglu et al. 1999). In our work, solutions were only added to a candidate list since we used the basic tabu search algorithm. But we can try using Pareto list such that the Pareto list collects the selected non-dominated solutions found by the algorithm. The candidate list, on the other hand, collects all other non-dominated solutions that were not selected as Pareto optimal solutions in an iteration. These solutions may become seed solutions if they maintain their non-dominated status in subsequent iterations. Using this process, the candidate list would give the opportunity to diversify the searching process.

Bibliography

Aiex, R. M., Binato, S. & Resende, M., 2003. Parallel GRASP with path relinking for job shop scheduling. *Parallel computing,* 29(4), pp. 393-430.

Aiex, R. M., Resende, M., Pardalos, P. M. & Toraldo, G., 2005. GRASP with path relinking for the three index assignment problem. *INFORMS Journal on computing,* 17(2), pp. 224-247.

Archambault, D., Munzner, T. & Auber, D., 2007. Topolayout: Multilevel graph layout by topological features. *IEEE transactions on visualization and computer graphics,* 13(2).

Archambault, D. & Purchase, H. C., 2013. The “map” in the mental map: Experimental results in dynamic graph drawing. *International journal of human-computer studies,* 71(11), pp. 1044-1055.

Balasubramanian, M. & Schwartz, E. L., 2002. The isomap algorithm and topological stability. *Science,* 295(5552), pp. 7-7.

Bandyopadhyay, S. & Saha, S., 2012. *Unsupervised classification: similarity measures, classical and metaheuristic approaches, and applications.* s.l.:Springer science & business media.

Barsky, A., Munzner, T., Gardy, J. & Kincaid, R., 2008. Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE transactions on visualization and computer graphics,* 14(6), pp. 1253-1260.

Batagelj, V. & Mrvar, A., 1998. Pajek-program for large network analysis. *Connections,* 21(2), pp. 47-57.

Batagelj, V. & Mrvar, A., 2006. *Pajek datasets. Web page http://vlado. fmf. uni-lj. si/pub/networks/data,* s.l.: s.n.

Batini, C., Nardelli, E. & Tamassia, R., 1986. A layout algorithm for data flow diagrams. *IEEE trans. software eng.,* Volume 4, pp. 538-546.

Baykasoglu, A., Owen, S. & Gindy, N., 1999. A Taboo search based approach to find the pareto optimal set in multiple objective optimization. *Engineering optimization,* 31(6), pp. 731-748.

Benlic, U. & Hao, J. K., 2011. An effective multilevel tabu search approach for balanced graph partitioning. *Computers & operations research,* 38(7), pp. 1066-1075.

Bertault, F., 1999. *A force-directed algorithm that preserves edge crossing properties.* Berlin, Heidelberg, Springer , pp. 351-358.

Bland, J. M. & Altman, D. G., 1995. Multiple significance tests: the Bonferroni method. *Bmj,* 310(6973), p. 170.

Blum, C. & Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR),* 35(3), pp. 268-308.

Blythe, J., McGrath, C. & Krackhardt, D., 1995. *The effect of graph layout on inference from social netword data.* Passau, Germany, Springer Berlin Heidelberg, pp. 40-51.

Böhringer, K. F. & Paulisch, F. N., 1990. *Using constraints to achieve stability in automatic graph layout algorithms.* s.l., In Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, pp. 43-51.

Bollobás, B., 1998. *Random graphs.* New York, NY, Springer, pp. 215-252.

Brandao, J. & Mercer, A., 1997. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European journal of operational research,* 100(1), pp. 180-191.

Branke, J., Bucher, F. & Schmeck, H., 1996. *Using genetic algorithms for drawing undirected graphs.* s.l., In the third nordic workshop on genetic algorithms and their applications.

Brank, J., 2004. *Drawing graphs using simulated annealing and gradient descent.* s.l., Zbornik C 7 mednarodne multi-konference Informacijska družba IS 2004.

Canuto, S. A., Resende, M. G. & Ribeiro, C. C., 2001. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks,* 38(1), pp. 50-58.

Carpano, M., 1980. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE trans. syst., man, and cybernetics,* 10(11), pp. 705-715.

Choe, Y., McCormick, B. H. & Koh, W., 2004. Network connectivity analysis on the temporally augmented C. elegans web: A pilot study. *In soc neurosci abstr,* 30(921.9).

Christensen, J., Marks, J. & Shieber, S., 1995. An empirical study of algorithms for point-feature label placement. *ACM transactions on graphics (TOG),* 14(3), pp. 203-232.

Chuzhoy, J., 2011. *An algorithm for the graph crossing number problem.* San Jose, CA, USA, In proceedings of the forty-third annual ACM symposium on Theory of computing, ACM, pp. 303-312.

Coello, C. C., Lamont, G. & Van Veldhuizen, D. A., 2006. *Evolutionary algorithms for solving multi-objective problems.* 2nd ed. New York, Inc. Secaucus, NJ, USA: Springer-Verlag.

Cohen, J., 1992. A power primer. *Psychological bulletin,* 112(1), p. 155.

Coleman, M. K. & Parker, D. S., 1996. Aesthetics-based graph layout for human consumption. *Software: practice and experience,* 26(12), pp. 1415-1438.

Coleman, T. F. & Moré, J. J., 1983. Estimation of sparse Jacobian matrices and graph coloring blems. *SIAM journal on numerical analysis,* 20(1), pp. 187-209.

Cordeau, J. F., Gendreau, M. & Laporte, G., 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks,* 30(2), pp. 105-119.

Cordeau, J. F. & Maischberger, M., 2012. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & operations research,* 39(9), pp. 2033-2050.

Daudin, J. J., Picard, F. & Robin, S., 2008. A mixture model for random graphs. *Statistics and computing,* 18(2), pp. 173-183.

Davidson, R. & Harel, D., 1996. Drawing graphs nicely using simulated annealing. *ACM transactions on graphics,* 15(4), pp. 301-331.

di Battista, G., Eades, P., Tamassia, R. & Tollis, I. G., 1999. *Graph drawing: algorithms for the visualisation of graphs.* s.l.:Prentice Hall.

Díaz, R. & Suárez, A. R., 2001. *A study of the capacity of the stochastic hill climbing to solve multi-objective problems.* s.l., In international symposium on adaptative system, pp. 37-40.

Dib, F. K. & Rodgers, P., 2014. A tabu search based approach for graph layout. *Journal of visual languages & computing,* 25(6), pp. 912-923.

Dib, F. K. & Rodgers, P., 2018. Graph drawing using tabu search coupled with path relinking. *PLoS ONE,* 13(5), p. e0197103.

Doerner, K. F. et al., 2006. Pareto ant colony optimization with ILP preprocessing in multiobjective project portfolio selection. *European journal of operational research,* 171(3), pp. 830-841.

Dogrusoz, U. et al., 2009. A layout algorithm for undirected compound graphs. *Information sciences,* 179(7), pp. 980-994.

Dogrusoz, U., Kakoulis, K. G., Madden, B. & Tollis, I. G., 2007. On labeling in graph visualization. *Information sciences,* 177(12), pp. 2459-2472.

Dorigo, M., Birattari, M. & Stutzle, T., 2006. Ant colony optimization. *IEEE computational intelligence magazine,* 1(4), pp. 28-39.

Dorigo, M. & Di Caro, G., 1999. Ant colony optimization: a new meta-heuristic. *In evolutionary computation, proceedings of the 1999 congress,* Volume 2, pp. 1470-1477.

Duarte, A., Laguna, M. & Marti, R., 2017. *Metaheuristics for business analytics: A decision modeling approach.* s.l.:Springer.

Dunn, O. J., 1961. Multiple comparisons among means. *Journal of the American statistical association,* 56(293), pp. 52-64.

Eades, P., 1984. A heuristic for graph drawing. *Congressus numeratum,* Volume 42, pp. 149-160.

Eades, P. & Feng, Q. W., 1996. *Multilevel visualization of clustered graphs.* Berlin, Heidelberg, Springer, pp. 101-112.

Eades, P. & Wormald, N. C., 1994. Edge crossings in drawings of bipartite graphs. *Algorithmica,* 11(4), pp. 379-403.

Eiben, A. E., Hinterding, R. & Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. *IEEE transactions on evolutionary computation,* 3(2), pp. 124-141.

Eloranta, T. & Mäkinen, E., 2001. TimGA: A genetic algorithm for drawing undirected graphs. *Divulgaciones matematicas,* 9(2), pp. 155-171.

Erdos, P. & Rényi, A., 1960. On the evolution of random graphs. *Publication of the mathematical institute of the Hungarian academy of sciences,* 5(1), pp. 17-60.

Escobar, J. W., Linfati, R. & Toth, P., 2013. A two-phase hybrid heuristic algorithm for the capacitated location-routing problem. *Computers & operations research,* 40(1), pp. 70-79.

Escobar, J. W., Linfati, R., Toth, P. & Baldoquin, M. G., 2014. A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. *Journal of heuristics,* 20(5), pp. 483-509.

Festa, P., Pardalos, P. M., Resende, M. G. & Ribeiro, C. C., 2002. Randomized heuristics for the MAX-CUT problem. *Optimization methods and software,* 17(6), pp. 1033-1058.

Fleurent, C. & Glover, F., 1999. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS journal on computing,* 11(2), pp. 198-204.

Flower, J., Rodgers, P. & Mutton, P., 2003. *Layout metrics for Euler diagrams.* s.l., In information visualization, 2003. IV 2003. proceedings. seventh international conference, IEEE, pp. 272-280.

Fonseca, C. M. & Fleming, P. J., 1993. *Genetic algorithms for multiobjective optimization: formulation discussion and generalization.* s.l., In ICGA, pp. 416-423.

Frick, A., Ludwig, A. & Mehldau, H., 1995. *A fast adaptive layout algorithm for undirected graphs.* London, UK, Proceedings of the DIMACS international workshop on graph drawing, Springer-Verlag, pp. 388--403.

Friden, C., Hertz, A. & de Werra, D., 1989. STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing, Springer-Verlag,* 42(1), pp. 35-44.

Fruchterman, T. M. & Reingold, E. M., 1991. Graph drawing by force-directed placement. *Software practice and experience,* 21(11), pp. 1129-1164.

Gambardella, L. M., Taillard, E. & Agazzi, G., 1999. *MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows.* London, UK, In new ideas in optimization, D. Corne et al., Eds. McGraw Hill, p. 63–76.

Gandibleux, X., Mezdaoui, N. & Fréville, A., 1997. A tabu search procedure to solve multiobjective combinatorial optimization problems. *In advances in multiple objective and goal programming, Springer Berlin Heidelberg,* pp. 291-300.

Gansner, E. R., Hu, Y. & North, S., 2013. A maxent-stress model for graph layout. *IEEE transactions on visualization and computer graphics,* 19(6), pp. 927-940.

Garey, M. R. & Johnson, D. S., 1983. Crossing number is NP-complete. *SIAM journal of algebraic and discrete methods,* 4(3), pp. 312-316.

Gendreau, M., Hertz, A. & Laporte, G., 1994. A tabu search heuristic for the vehicle routing problem. *Management science,* 40(10), pp. 1276-1290.

Gendreau, M. & Potvin, J. Y., 2014. Tabu search. In: *Search methodologies.* US: Springer, pp. 243-263.

Ghamlouche, I., Crainic, T. G. & Gendreau, M., 2004. Path relinking, cycle based neighbourhoods and capacitated multicommodity network design. *Annals of operations research,* 131(1-4), pp. 109-133.

Gibson, H., Faith, J. & Vickers, P., 2013. A survey of two-dimensional graph layout techniques for information visualisation. *Information visualization,* 12((3-4)), pp. 324-357.

Girvan, M. & Newman, M. E., 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences,* 99(12), pp. 7821-7826.

Gleiser, P. M. & Danon, L., 2003. Community structure in jazz. *Advances in complex systems,* 6(04), pp. 565-573.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. *Computer and operations research,* 13(5), pp. 533-549.

Glover, F., 1989. Tabu search - part I. *ORSA journal on computing,* 1(3), pp. 190-206.

Glover, F., 1990. Tabu search - part II. *ORSA journal on computing,* 2(1), pp. 4-32.

Glover, F., 1997. *Tabu search and adaptive memory programming—advances, applications and challenges.* s.l., In interfaces in computer science and operations research, Springer US, pp. 1-75.

Glover, F. & Greenberg, H. J., 1989. New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European journal of operational research,* Volume 39, pp. 119-130.

Glover, F. & Laguna, M., 1997. *Tabu search.* Boston: Kluwer academic publishers.

Glover, F., Laguna, M. & Martí, R., 2000. Fundamentals of scatter search and path relinking. *Control and cybernetics,* 29(3), pp. 653-684.

Grandinetti, L., Guerriero, F., Laganà, D. & Pisacane, O., 2012. An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem. *Computers & operations research,* 39(10), pp. 2300-2309.

Grodzevich, O. & Romanko, O., 2006. *Normalization and other topics in multi-objective optimization,* s.l.: Proceedings of the fields-MITACS industrial problems workshop.

Hachul, S. & Jünger, M., 2004. *Drawing large graphs with a potential-field-based multilevel algorithm.* Berlin, Heidelberg, Springer, pp. 285-295.

Hansen, M., 1997. *Tabu search for multiobjective optimization: MOTS.* s.l., In proceedings of the 13th international conference on multiple criteria decision making.

Hawkins, D. M., 2004. The problem of overfitting. *Journal of chemical information and computer sciences,* 44(1), pp. 1-12.

Hertz, A. & De Werra, D., 1989. Using tabu search techniques for graph coloring. *Computing,* 39(4), pp. 345-351.

Hertz, A., Taillard, E. & De Werra, D., 1995. A tutorial on tabu search. *In proceedings of giornate di lavoro AIRO,* Volume 95, pp. 13-24.

He, W. & Marriott, K., 1998. Constrained graph layout. *Constraints,* 3(4), pp. 289-314.

Holm, S., 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics,* 6(2), pp. 65-70.

Ho, S. C. & Gendreau, M., 2006. Path relinking for the vehicle routing problem. *Journal of heuristics,* 12(1-2), pp. 55-72.

Huang, X., Lai, W., Sajeev, A. S. M. & Gao, J., 2007. A new algorithm for removing node overlapping in graph visualization. *Information sciences,* 177(14), pp. 2821-2844.

Iredi, S., Merkle, D. & Middendorf, M., 2001. Bi-criterion optimization with multi colony ant algorithms. *In evolutionary multi-criterion optimization, Springer Berlin/Heidelberg,* pp. 359-372.

Jacomy, M., Venturini, T., Heymann, S. & Bastian, M., 2014. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PloS ONE,* 9(6), p. e98679.

Jancauskas, V., Kaukas, G., Zilinskas, A. & Zilinskas, J., 2012. *On multi-objective optimization aided visualization of graphs related to business process diagrams.* s.l., DB & local proceedings, pp. 71-80.

Jayalakshmi, T. & Santhakumaran, A., 2011. Statistical normalization and back propagation for classification. *International journal of computer theory and engineering,* 3(1), pp. 1793-8201.

Kamada, T. & Kawai, S., 1989. An algorithm for drawing general undirected graphs. *Journal of information processing letters,* 31(1), pp. 7-15.

Kim, I. Y. & de Weck, O. L., 2005. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and multidisciplinary optimization,* 29(2), pp. 149-158.

Kleywegt, A. J. & Shapiro, A., 2001. Stochastic optimization. In: *Handbook of Industrial Engineering.* s.l.:John Wiley & Sons, pp. 2625-2649.

Konak, A., Coit, D. W. & Smith, A. E., 2006. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability engineering & system safety,* 91(9), pp. 992-1007.

Kosak, C., Marks, J. & Shieber, S., 1991. *A parallel genetic algorithm for network diagram layout.* San Diego, CA, USA, Proceedings of the 4th international conference on genetic algorithms, pp. 458-465.

Kosak, C., Marks, J. & Shieber, S., 1994. Automating the layout of network diagrams with specified visual organization. *IEEE transactions on systems, man, and cybernetics,* 24(3), pp. 440-454.

Kotsiantis, S. B., Kanellopoulos, D. & Pintelas, P. E., 2006. Data preprocessing for supervised leaning. *International journal of computer science,* 1(2), pp. 111-117.

Krebs, V., n.d. *http://www.orgnet.com,* s.l.: unpublished.

Laguna, M. & Marti, R., 1999. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS journal on computing A.,* 11(1), pp. 44-52.

Laguna, M., Marti, R. & Valls, V., 1997. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers and operations research,* 24(12), pp. 1175-1186.

Leighton, T. & Rao, S., 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM),* 46(6), pp. 787-832.

Li, H. & Landa-Silva, D., 2011. An adaptive evolutionary multi-objective approach based on simulated annealing. *Evolutionary computation,* 19(4), pp. 561-595.

Lim, A. & Chee, Y. M., 1991. Graph partitioning using tabu search. *IEEE international symposium on circuits and systems,* Volume 2, pp. 1164-1167.

Lin, C. C., Lee, Y. Y. & Yen, H. C., 2011. Mental map preserving graph drawing using simulated annealing. *Information sciences,* 181(19), pp. 4253-4272.

López-Ibánez, M., Paquete, L. & Stutzle, T., 2004. On the design of ACO for the biobjective quadratic assignment problem. *Lecture notes in computer science,* Volume 3172, pp. 214-225.

Lusseau, D. et al., 2003. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behavioral ecology and sociobiology,* 54(4), pp. 396-405.

Maaten, L. V. D. & Hinton, G., 2008. Visualizing data using t-SNE. *Journal of machine learning research,* 9(Nov), pp. 2579-2605.

Mäkinen, E. & Sieranta, M., 1994. Genetic algorithms for drawing bipartite graphs. *International journal of computer mathematics,* 53(3-4), pp. 157-166.

Marti, R., 1998. A tabu search algorithm for the bipartite drawing problem. *European journal of operational research,* 106(2), pp. 558-569.

Martí, R., Campos, V., Resende, M. G. & Duarte, A., 2015. Multiobjective GRASP with path relinking. *European journal of operational research,* 240(1), pp. 54-71.

Martí, R. & Laguna, M., 2003. Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete applied mathematics,* 127(3), pp. 665-678.

Melián, C. J. & Bascompte, J., 2004. Food web cohesion. *Ecology,* 85(2), pp. 352-358.

Metropolis, N. et al., 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics,* 21(6), pp. 1087-1092.

Miller, Z. & Orlin, J. B., 1985. NP-completeness for minimizing maximum edge length in grid embeddings. *Journal of algorithms,* 6(1), pp. 10-16.

Misue, K., Eades, P., Lai, W. & Sugiyama, K., 1995. Layout adjustment and the mental map. *Journal of visual languages & computing,* 6(2), pp. 183-210.

Murata, T., Ishibuchi, H. & Tanaka, H., 1996. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & industrial engineering,* 30(4), pp. 957-968.

Nam, D. & Park, C. H., 2000. Multiobjective simulated annealing: A comparative study to evolutionary algorithms. *International journal of fuzzy systems,* 2(2), pp. 87-97.

Newman, M. E., 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3), p. 036104.

Noack, A., 2007. Energy models for graph clustering. *Journal of graph algorithms applications*, 11(2), pp. 453-480.

Oliveira, C. A., Pardalos, P. M. & Resende, M., 2004. GRASP with path-relinking for the quadratic assignment problem. *In experimental and efficient algorithms. Springer Berlin Heidelberg*, pp. 356-368.

Ortmann, M., Klimenta, M. & Brandes, U., 2016. *A sparse stress model.* In international symposium on graph drawing and network visualization, Springer international publishing, pp. 18-32.

Osman, I. H., 2006. A tabu search procedure based on a random Roulette diversification for the weighted maximal planar graph problem. *Computers & operations research,* 33(9), pp. 2526-2546.

Pacheco, J. & Marti, R., 2006. Tabu search for a multi-objective routing problem. *Journal of the operational research society*, 57(1), pp. 29-37.

Parragh, S. N., Doerner, K. F. & Hartl, R. F., 2008. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft,* 58(1), pp. 21-51.

Peng, B., Lu, Z. & Cheng, T., 2014. *A tabu search/path relinking algorithm to solve the job shop scheduling problem.* s.l.:arXiv preprint arXiv:1402.5613.

Perneger, T. V., 1998. What's wrong with Bonferroni adjustments.. *British medical journal*, 316(7139), p. 1236.

Purchase, H. C., 1997. Which aesthetic has the greatest effect on human understanding?. *Proceedings of graph drawing symposium*, pp. 248-261.

Purchase, H. C., 2002. Metrics for graph drawing aesthetics. *Journal of visual languages and computing,* 13(5), pp. 501-516.

Purchase, H. C., Cohen, R. F. & Jones, M., 1996. *Validating graph drawing aesthetics.* s.l., In graph drawing, Springer Berlin Heidelberg, pp. 435-446.

Rahimi-Vahed, A., Crainic, T. G., Gendreau, M. & Rei, W., 2013. A path relinking algorithm for a multi-depot periodic vehicle routing problem. *Journal of heuristics,* 19(3), pp. 497-524.

Resende, M. G., Martí, R., Gallego, M. & Duarte, A., 2010. GRASP and path relinking for the max–min diversity problem. *Computers & operations research,* 37(3), pp. 498-508.

Resende, M. G. & Werneck, R. F., 2004. A hybrid heuristic for the p-median problem. *Journal of heuristics,* 10(1), pp. 59-88.

Resende, M. & Ribeiro, C., 2003. A GRASP with path-relinking for private virtual circuit routing. *Networks* , 41(2), pp. 104-114.

Ribeiro, C. C. & Resende, M. G., 2012. Path-relinking intensification methods for stochastic local search algorithms. *Journal of heuristics,* 18(2), pp. 193-214.

Ribeiro, C. C. & Vianna, D. S., 2009. A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *International transactions in operational research,* 16(5), pp. 641-657.

Rolland, E., Pirkul, H. & Glover, F., 1996. Tabu search for graph partitioning. *Annals of operations research ,* 63(2), pp. 209-232.

Rosete-Suárez, A., Ochoa-Rodrıguez, A. & Sebag, M., 1999. Automatic graph drawing and stochastic hill climbing. *In proceedings of the genetic and evolutionary computation conference,* Volume 2, pp. 1699-1706.

Rowe, L. A. et al., 1987. A browser for directed graphs. *Software: practice and experience,* 17(1), pp. 61-76.

Sánchez-Oro, J. & Duarte, A., 2012. Grasp with path relinking for the sumcut problem. *International journal of combinatorial optimization problems and informatics,* 3(1), pp. 3-11.

Scaparra, M. P. & Church, R. L., 2005. A GRASP and path relinking heuristic for rural road network development. *Journal of heuristics,* 11(1), pp. 89-108.

Shalabi, L. A., Shaaban, Z. & Kasasbeh, B., 2006. Data mining: A preprocessing engine. *Journal of computer science,* 2(9), pp. 735-739.

Shapiro, S. S. & Wilk, M. B., 1965. An analysis of variance test for normality (complete samples). *Biometrika,* 52(3/4), pp. 591-611.

Simonetto, P., Archambault, D., Auber, D. & Bourqui, R., 2011. *ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges.* Oxford, UK, Blackwell Publishing Ltd., pp. 1071-1080.

Smith, K. I. et al., 2008. Dominance-based multiobjective simulated annealing. *IEEE transactions on evolutionary computation,* 12(3), pp. 323-342.

Souza, M. C., Duhamel, C. & Ribeiro, C., 2004. A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. *In metaheuristics: computer decision-making. Springer US,* pp. 627-657.

Stott, J., Rodgers, P., Martinez-Ovando, J. C. & Walker, S. G., 2011. Automatic metro map layout using multicriteria optimization. *IEEE transactions on visualization and computer graphics,* 17(1), pp. 101-114.

Sugiyama, K., Tagawa, S. & Toda, M., 1981. Methods for visual understanding of hierarchical system structures. *IEEE trans, syst., man, and cybernetics,* 11(2), pp. 109-125.

Suman, B. & Kumar, P., 2006. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the operational research society,* 57(10), pp. 1143-1160.

Sunar, M. & Kahraman, R., 2001. A comparative study of multiobjective optimization methods in structural design. *Turkish journal of engineering and environmental sciences,* 25(2), pp. 69-78.

Talbi, E. G. & Muntean, T., 1993. *Hill-climbing, simulated annealing and genetic algorithms: a comparative study and application to the mapping problem.* s.l., Proceeding of the twenty-sixth Hawaii international conference, IEEE, pp. 565-573.

Tamassia, R. (., 2013. *Handbook of graph drawing and visualization.* s.l.:CRC press.

Tamassia, R., Battista, G. & Batini, C., 1988. Automatic graph drawing and readability of diagrams. *IEEE trans. syst., man, and cybernetics*, 18(1), pp. 61-79.

Thakur, T. & Dhiman, J., 2011. A tabu search algorithm for multi-objective purpose of feeder reconfiguration. *Journal of electrical and electronics engineering research,* 3(4), pp. 71-79.

Titiloye, O. & Crispin, A., 2012. Parameter tuning patterns for random graph coloring with quantum annealing. *PloS one,* 7(11), p. e50060.

Ulungu, L. E., Teghem, J. & Ost, C., 1998. Interactive simulated annealing in a multiobjective framework: application to an industrial problem. *Journal of operational research society,* 49(10), pp. 1044-1050.

Upton, G. & Cook, I., 2014. *A dictionary of statistics.* 3 ed. s.l.:Oxford university press.

Valls, V., Martí, R. & Lino, P., 1996. A tabu thresholding algorithm for arc crossing minimization in bipartite graphs. *Annals of operations research,* 63(2), pp. 233-251.

Van Ham, F. & Van Wijk, J. J., 2004. *Interactive visualization of small world graphs.* Austin, Texas, Proc. IEEE symposium on information visualization, IEEE CS Press, pp. 199-206.

Vrajitoru, D., 2009. *Multiobjective genetic algorithm for a graph drawing problem.* s.l., In proceedings of the midwest artificial intelligence and cognitive science conference, pp. 28-43.

Walshaw, C., 2000. *A multilevel algorithm for force-directed graph drawing.* Berlin, Heidelberg, Springer, pp. 171-182.

Ware, J. M., Jones, C. B. & Thomas, N., 2003. Automated map generalization with multiple operators: a simulated annealing approach. *International journal of geographical information science,* 17(8), pp. 743-769.

Ware, M. & Richards, N., 2013. *An ant colony system algorithm for automatically schematizing transport network data sets.* s.l., In evolutionary computation (CEC), 2013 IEEE congress, pp. 1892-1900.

White, J. G., Southgate, E., Thomson, J. N. & Brenner, S., 1986. The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical transactions of the royal society of London. B, biological sciences,* 314(1165), pp. 1-340.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. *Biometrics bulletin,* 1(6), pp. 80-83.

Yıldız, A. R., 2009. An effective hybrid immune-hill climbing optimization approach for solving design and manufacturing optimization problems in industry. *Journal of materials processing technology,* 209(6), pp. 2773-2780.

Zachary, W., 1977. An information flow modelfor conflict and fission in small groups1. *Journal of anthropological research,* 33(4), pp. 452-473.

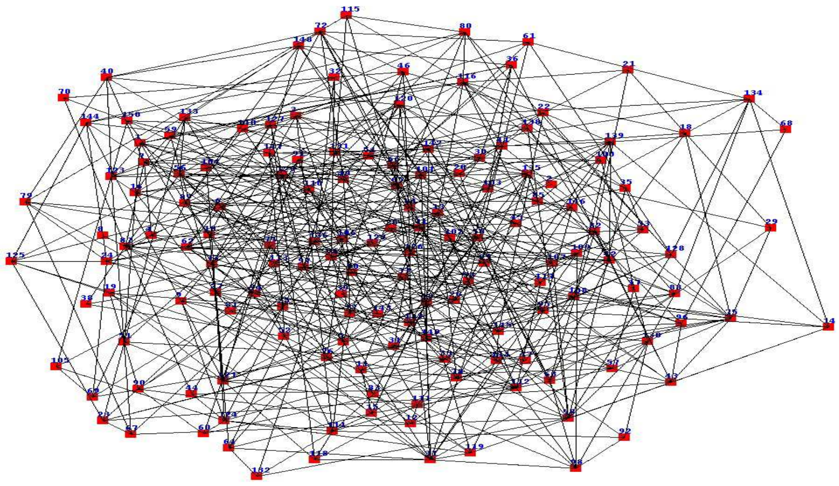
Appendix A Sample Layouts from Hill Climbing, Simulated Annealing, and Our Proposed Tabu Search-based Algorithm

In this appendix, we show sample layouts, from the graph datasets described in Table 5.1 and Table 5.2, generated by the three drawing algorithms discussed in Chapter 5: hill climbing, simulated annealing, and our proposed tabu search-based algorithm. Note that, the fitness function includes the measures of the following aesthetics: node-node occlusion, edge length, edge crossings, and angular resolution.

A.1 Sample Layouts from Datasets of Table 5.1

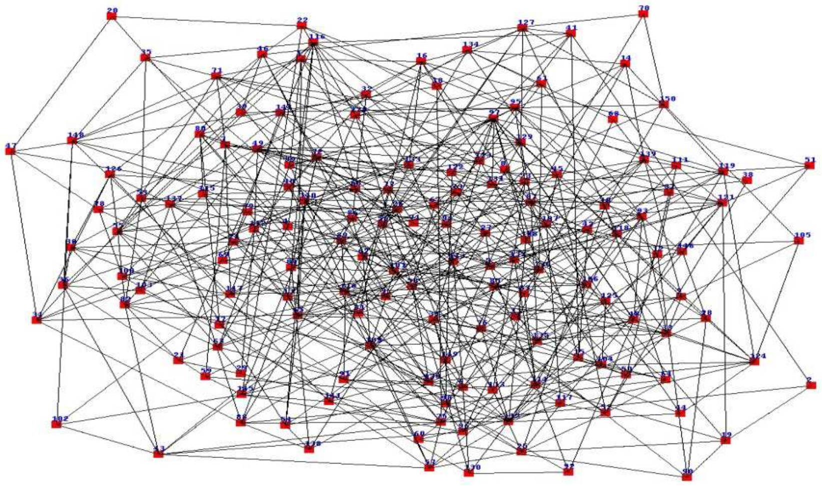
Hill climbing

Fitness = 0.612



Simulated annealing

Fitness = 0.490



Tabu search

Fitness = 0.509

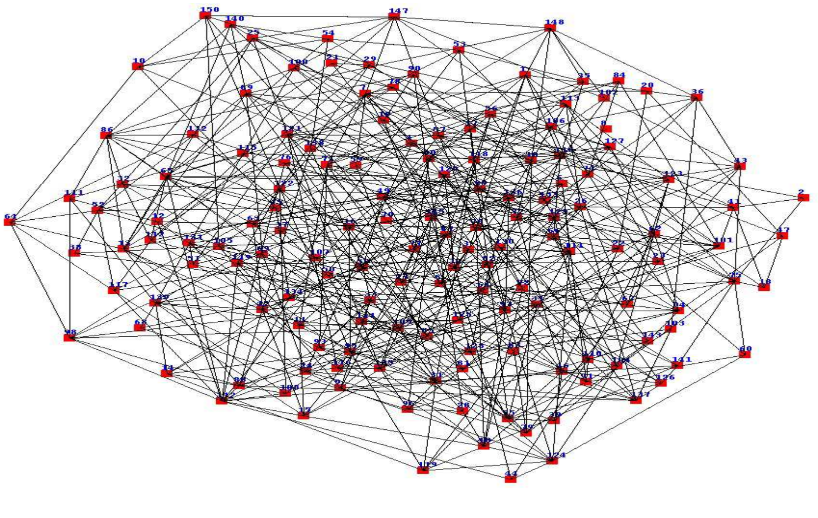
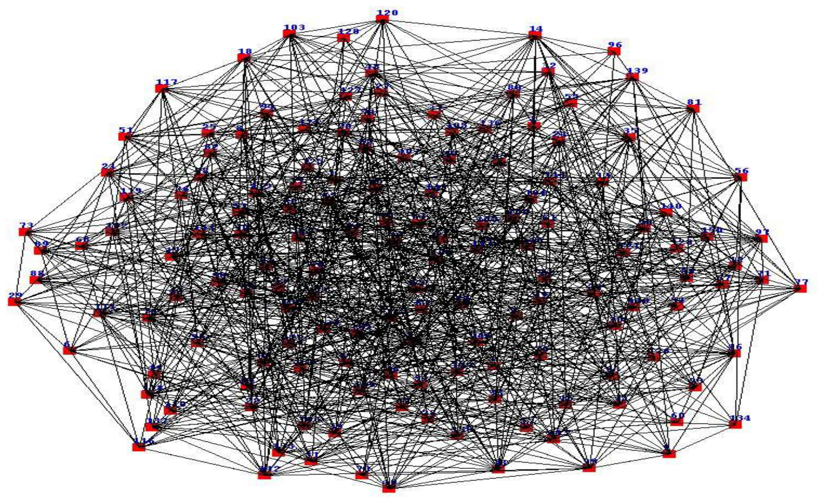


Figure A.1.1 Sample layouts from group 1A in Table 5.1

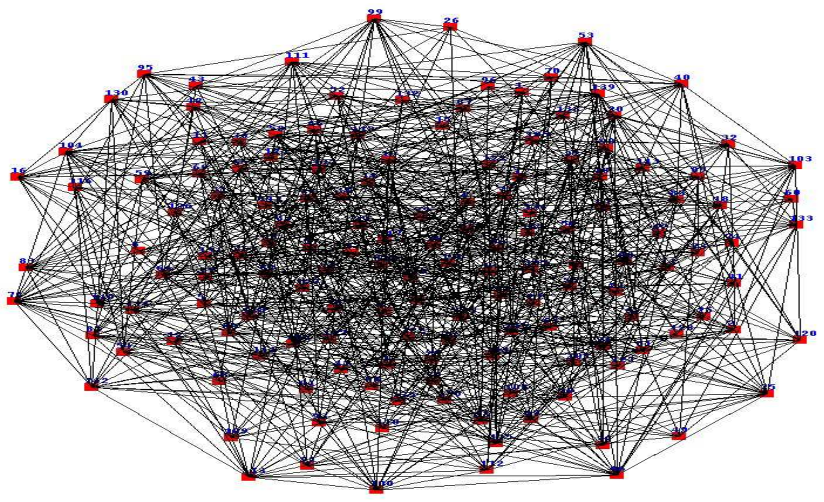
Hill climbing

Fitness = 0.859



Simulated annealing

Fitness = 0.752



Tabu search

Fitness = 0.825

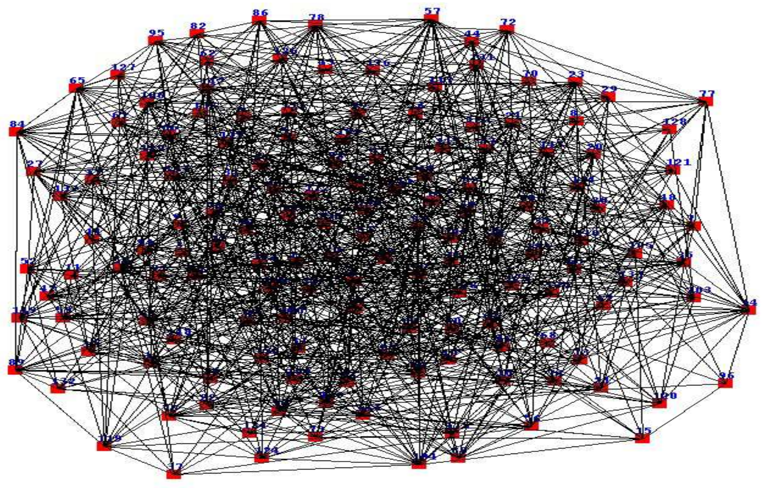
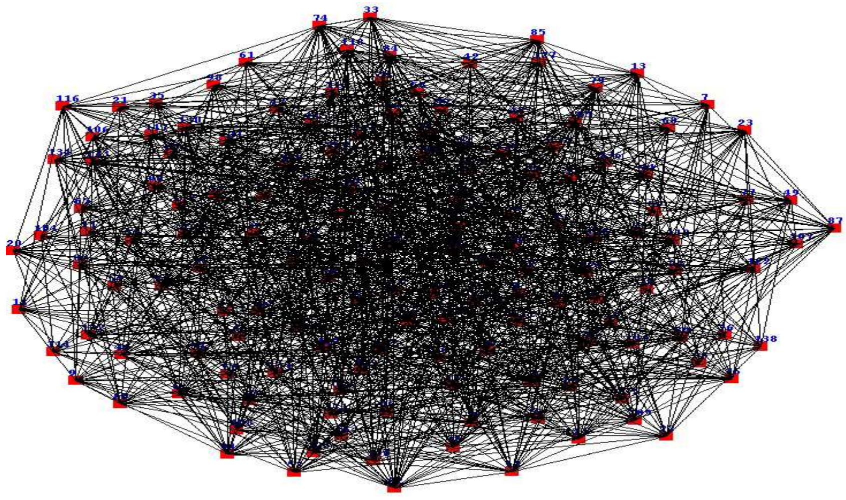


Figure A.1.2 Sample layouts from group 2A in Table 5.1

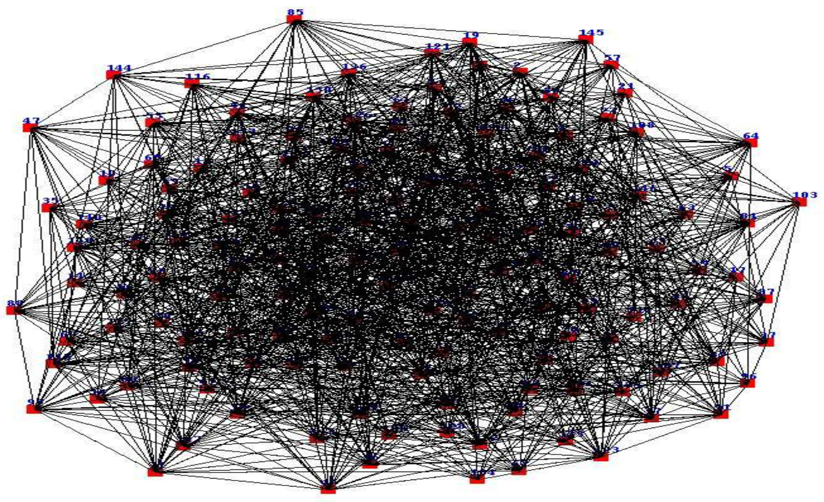
Hill climbing

Fitness = 0.925



Simulated annealing

Fitness = 0.879



Tabu search

Fitness = 0.889

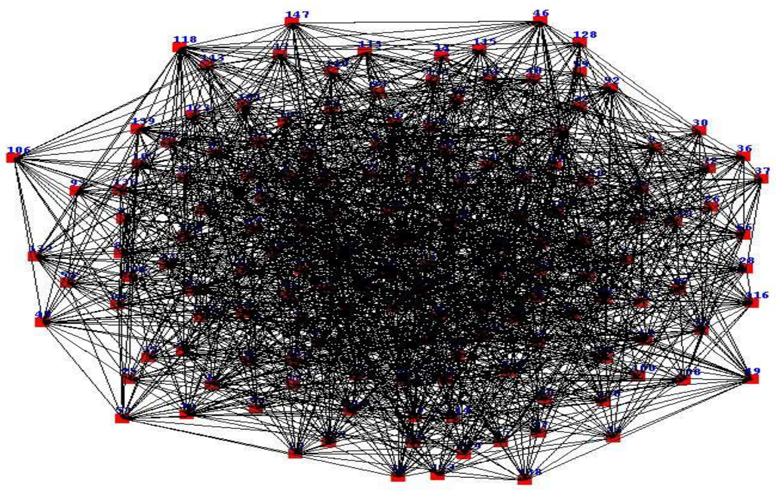
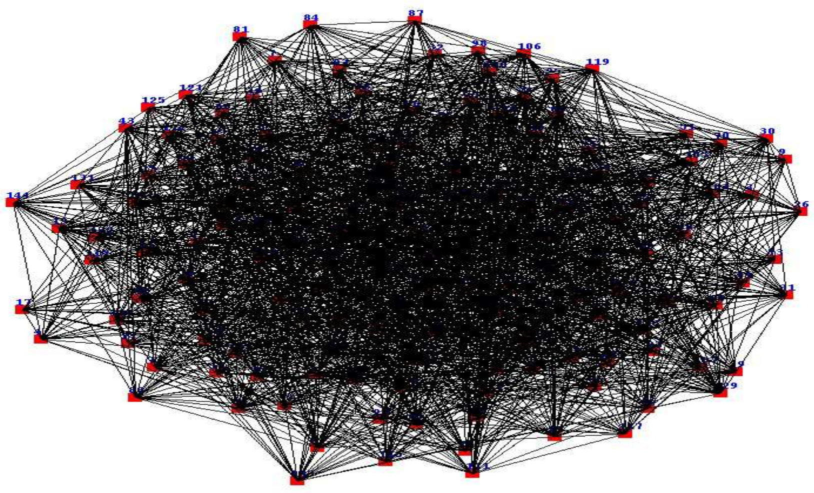


Figure A.1.3 Sample layouts from group 3A in Table 5.1

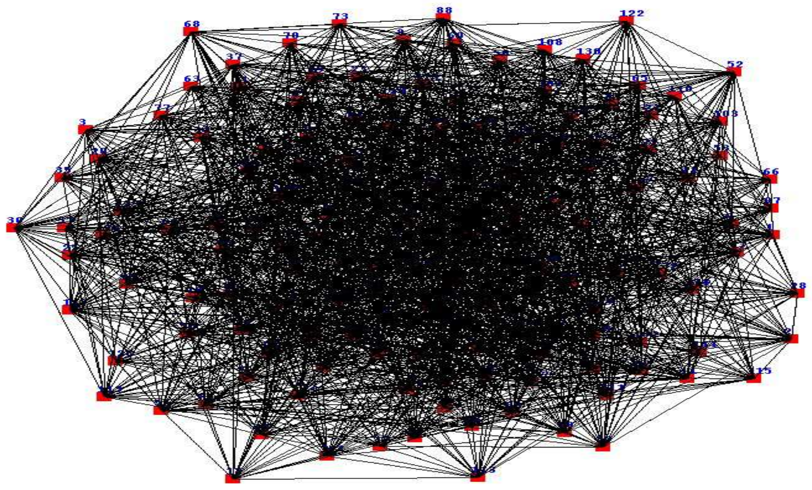
Hill climbing

Fitness = 1.038



Simulated annealing

Fitness = 0.910



Tabu search

Fitness = 0.993

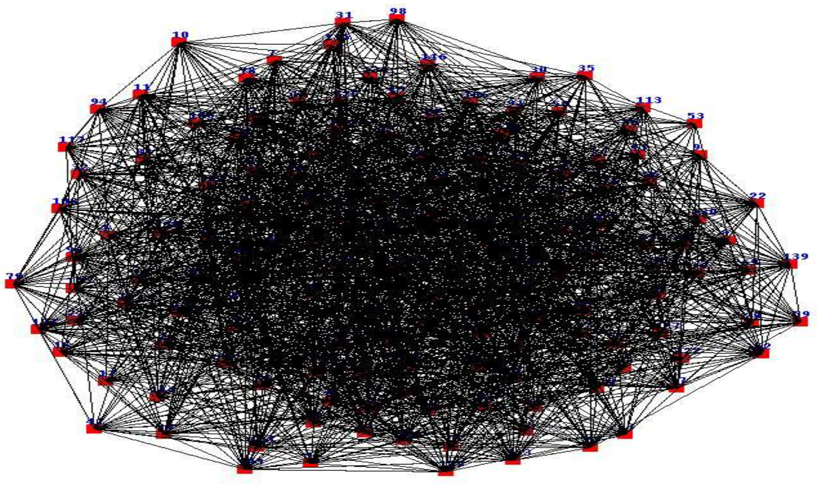
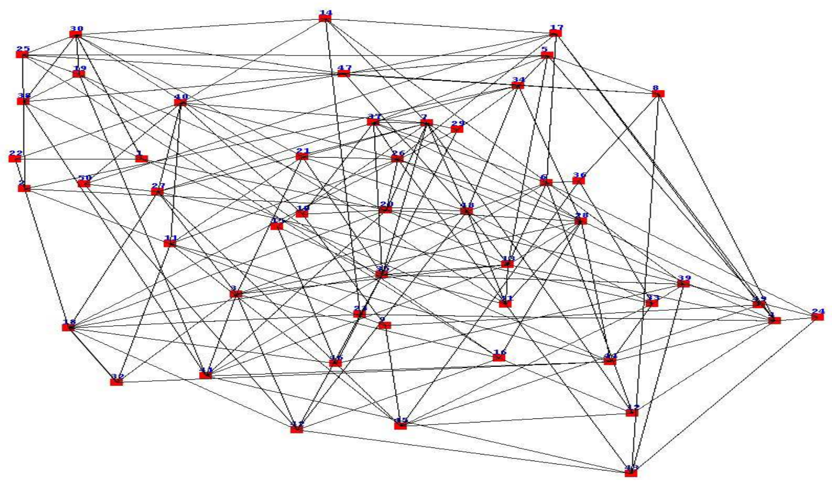


Figure A.1.4 Sample layouts from group 4A in Table 5.1

A.2 Sample Layouts from Datasets of Table 5.2

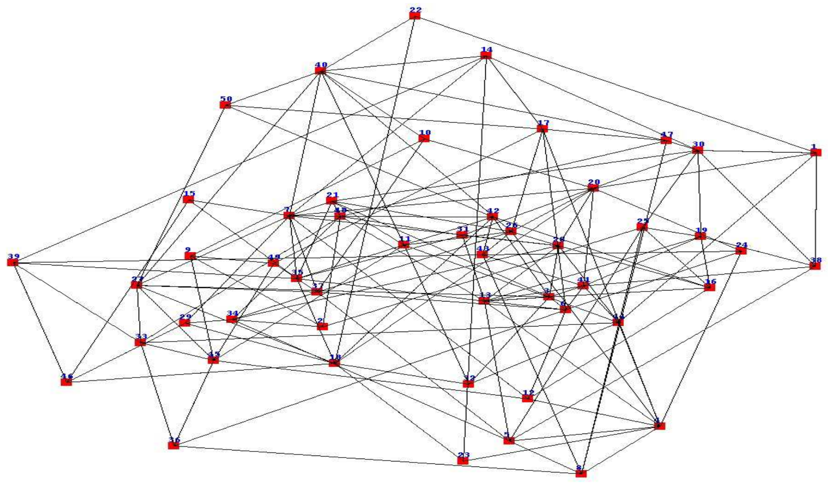
Hill climbing

Fitness = 0.417



Simulated annealing

Fitness = 0.276



Tabu search

Fitness = 0.327

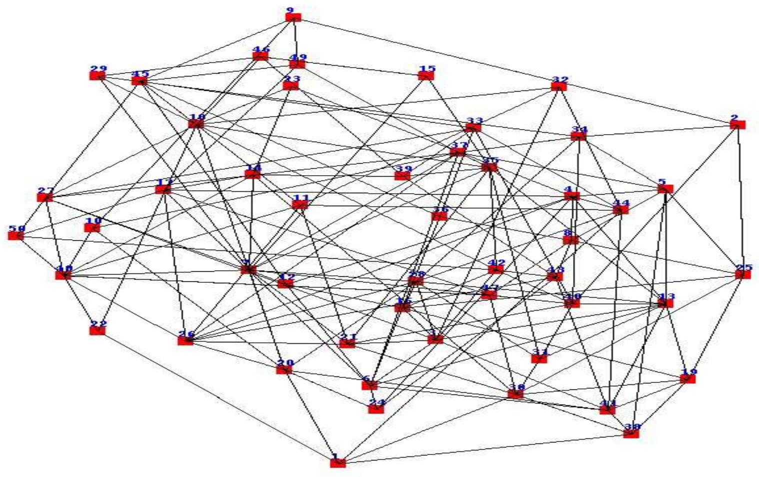


Figure A.2.1 Sample layouts from group 1B in Table 5.2

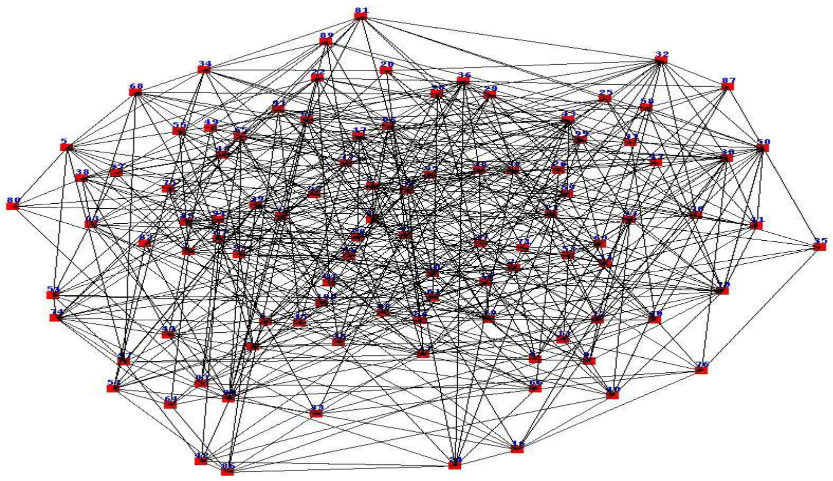
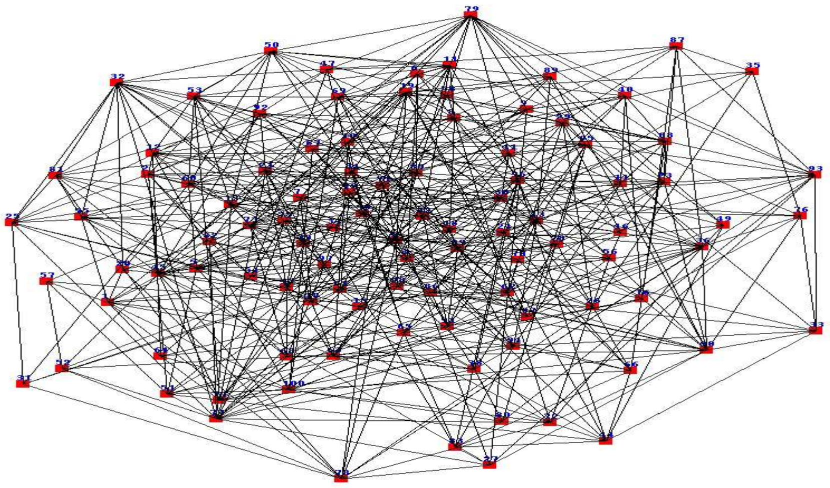
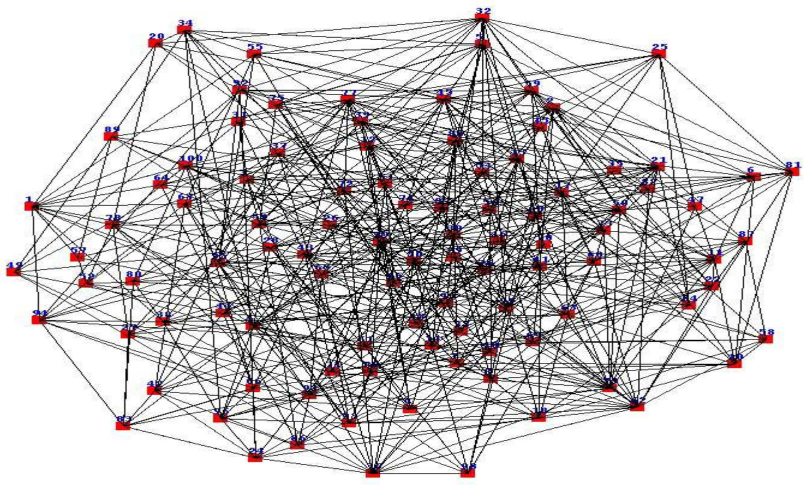
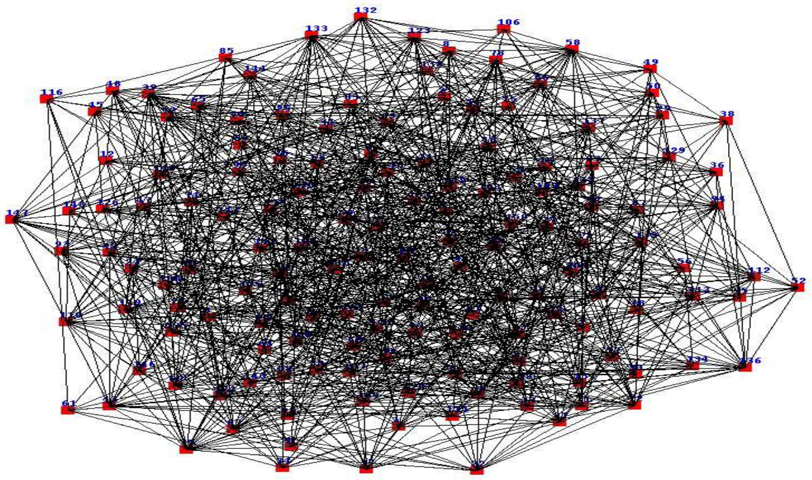
Hill climbing **Fitness = 0.632**	
Simulated annealing **Fitness = 0.560**	
Tabu search **Fitness = 0.612**	

Figure A.2.2 Sample layouts from group 2B in Table 5.2

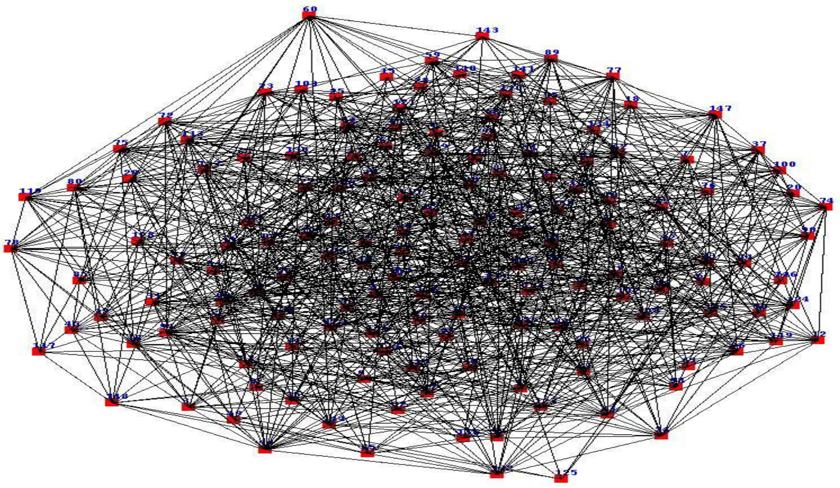
Hill climbing

Fitness = 1.249



Simulated annealing

Fitness = 0.760



Tabu search

Fitness = 0.858

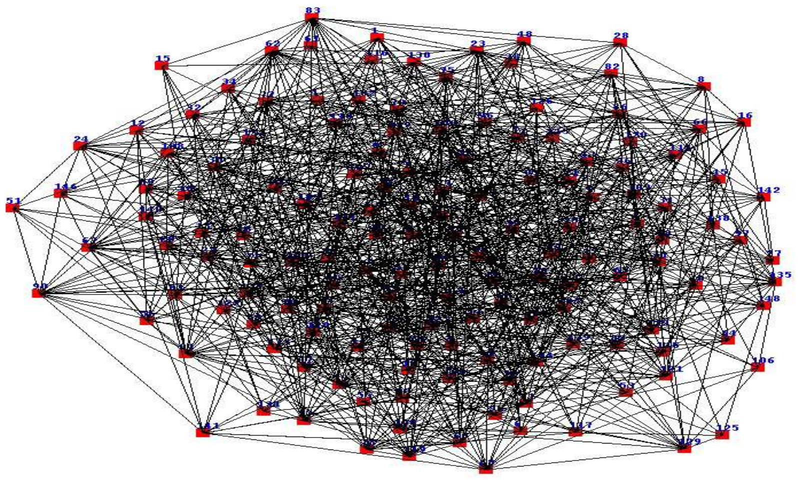
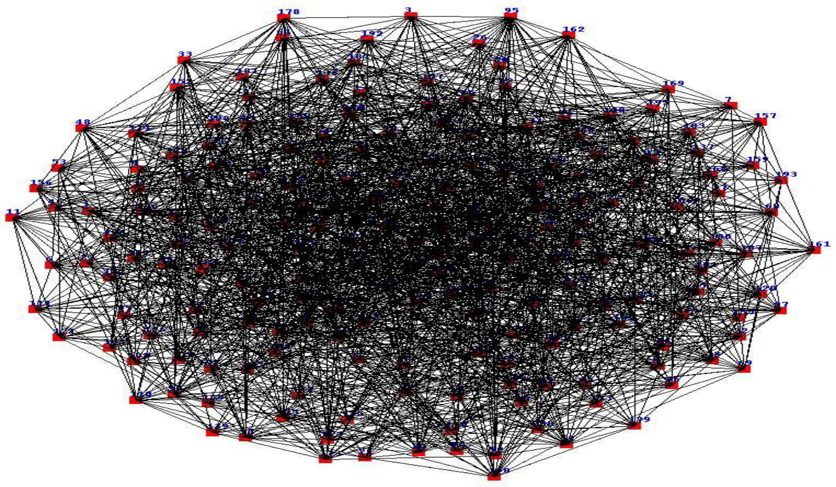


Figure A.2.3 Sample layouts from group 3B in Table 5.2

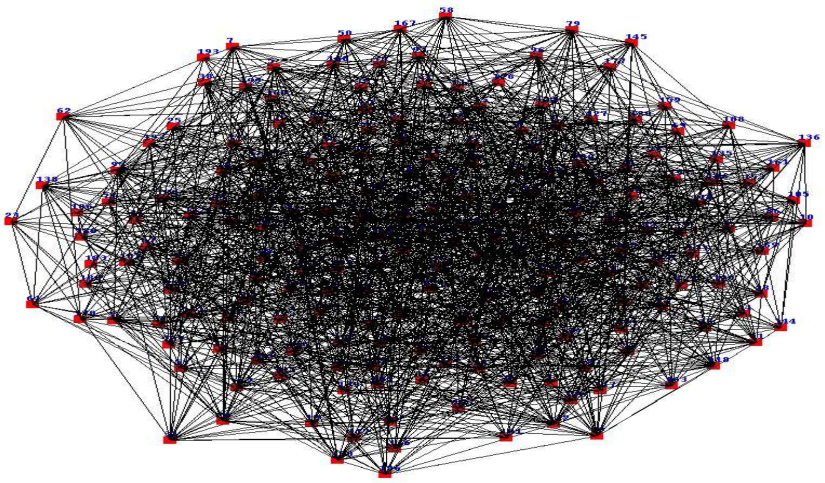
Hill climbing

Fitness = 1.068



Simulated annealing

Fitness = 0.966



Tabu search

Fitness = 1.034

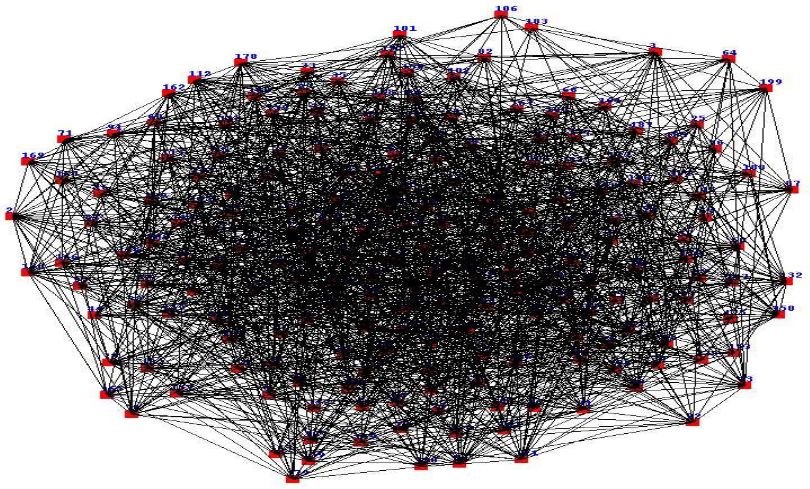


Figure A.2.4 Sample layouts from group 4B in Table 5.2

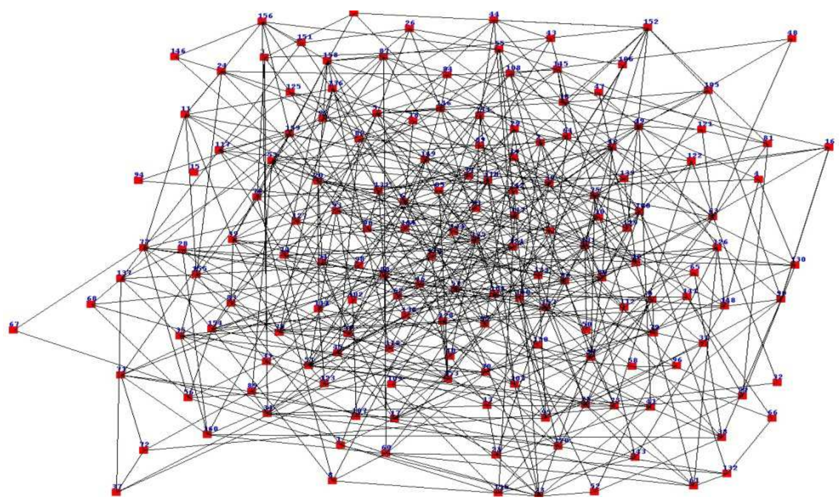
Appendix B Sample Layouts from Simulated Annealing, Our Proposed Tabu Search-based Algorithm, and Path Relinking Coupled with Tabu Search

In this appendix, we show sample layouts, from the graph datasets described in Table 7.1 and Table 7.2, generated by the three drawing algorithms discussed in Chapter 7: simulated annealing, our proposed tabu search-based algorithm, and path relinking coupled with tabu search. Note that, the fitness function includes the measures of the following aesthetics: node-node occlusion, edge length, edge crossings, and angular resolution.

B.1 Sample Layouts from Datasets of Table 7.1

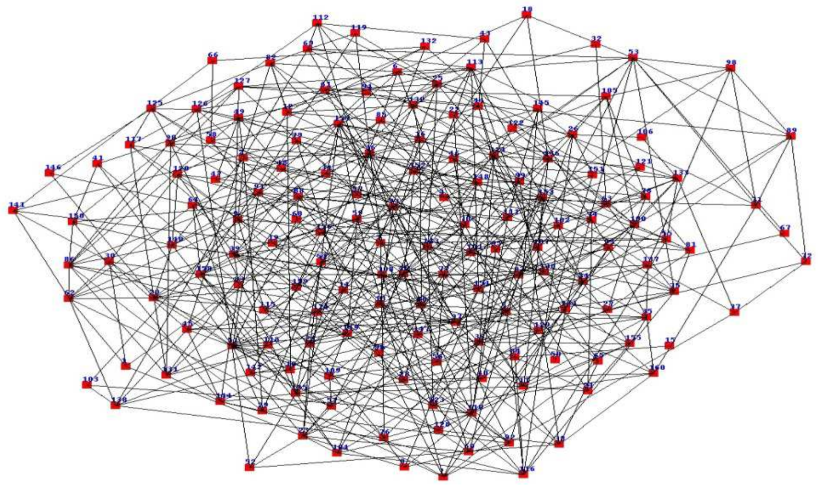
Simulated annealing

Fitness = 0.489



Tabu search

Fitness = 0.497



Path Relinking +

Tabu search

Fitness = 0.238

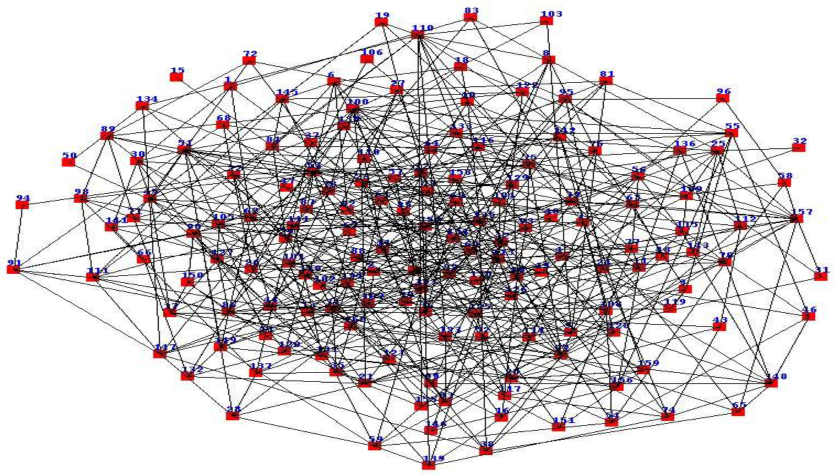
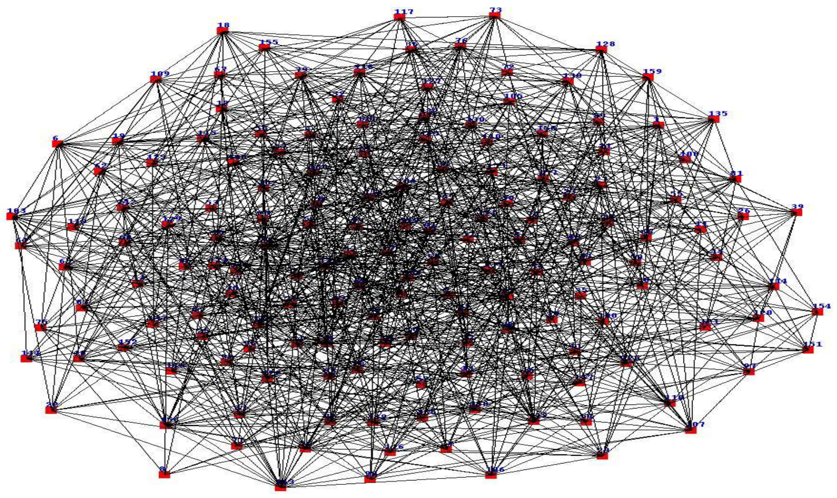


Figure B.1.1 Sample layouts from group 1C in Table 7.1

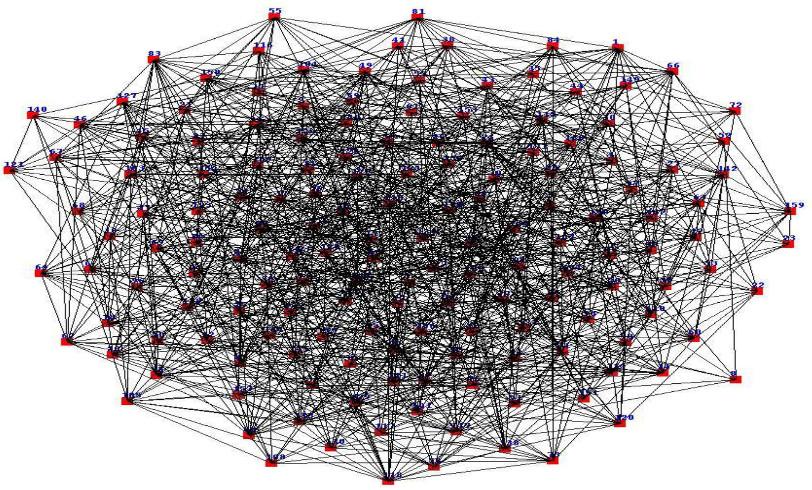
Simulated annealing

Fitness = 0.766



Tabu search

Fitness = 0.835



Path Relinking + Tabu search

Fitness = 0.398

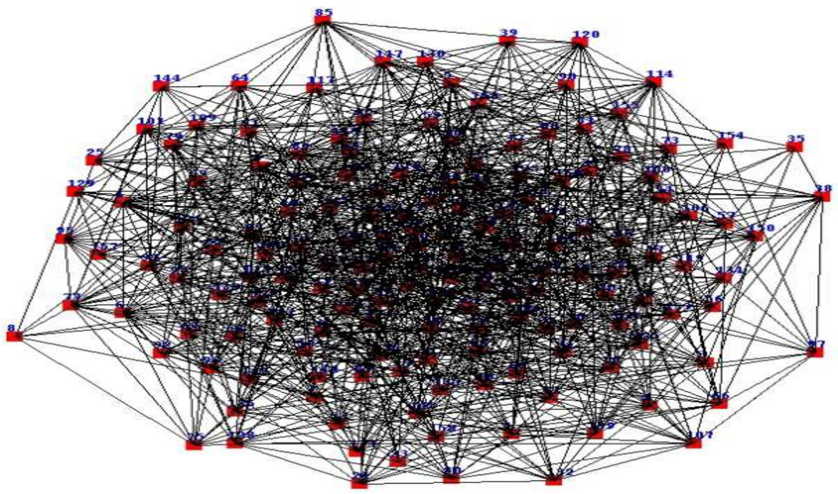
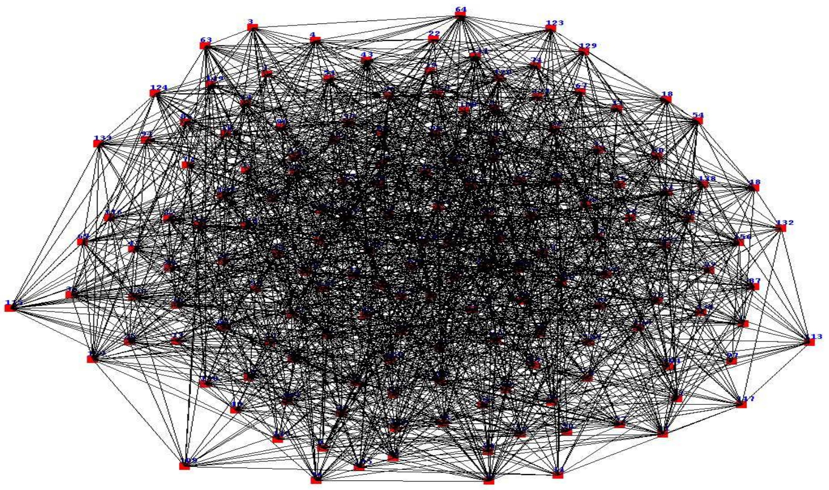


Figure B.1.2 Sample layouts from group 2C in Table 7.1

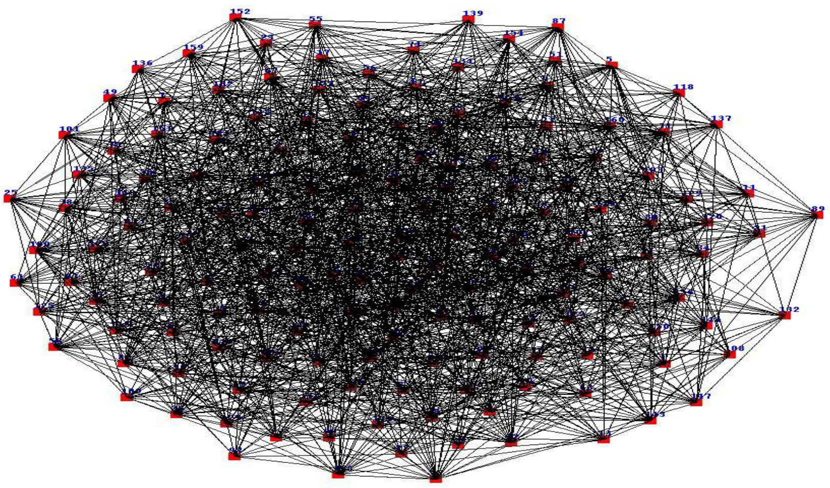
Simulated annealing

Fitness = 0.911



Tabu search

Fitness = 0.930



Path Relinking +
Tabu search

Fitness = 0.481

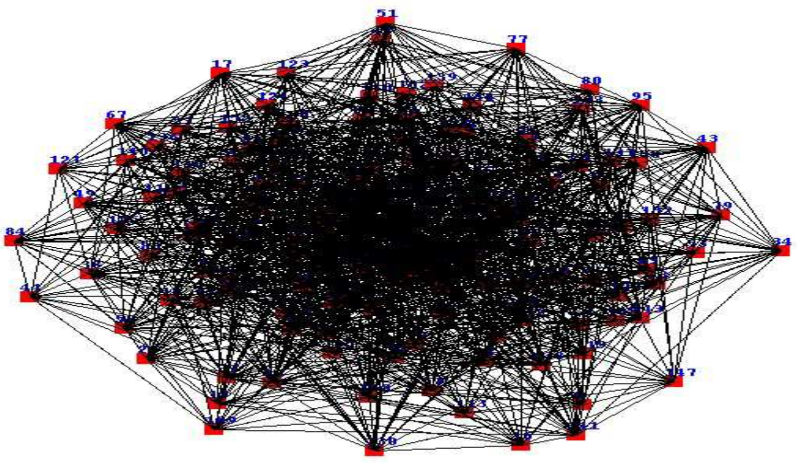


Figure B.1.3 Sample layouts from group 3C in Table 7.1

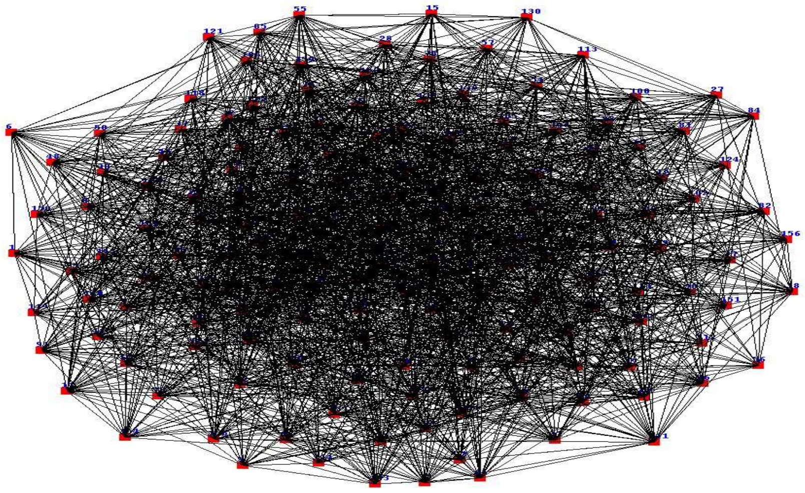
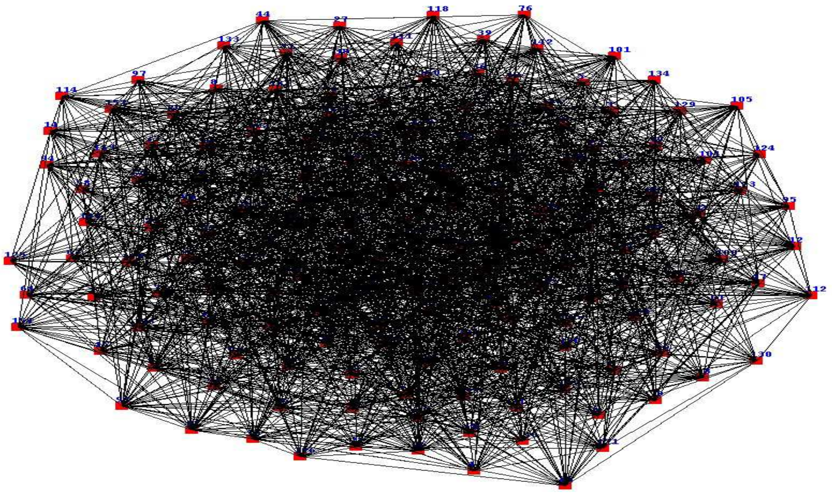
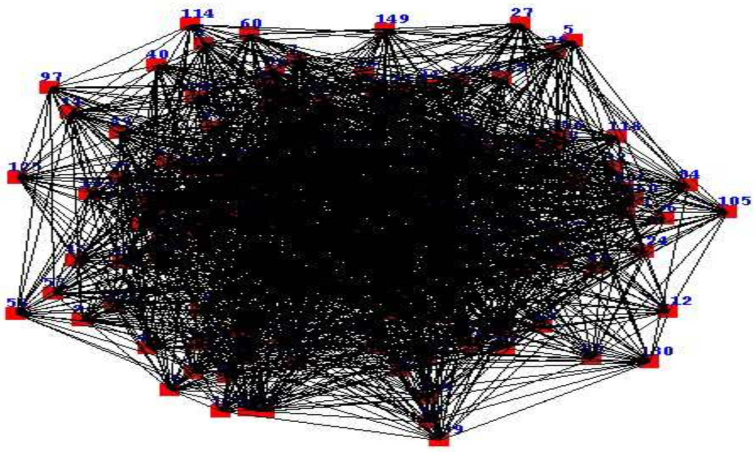
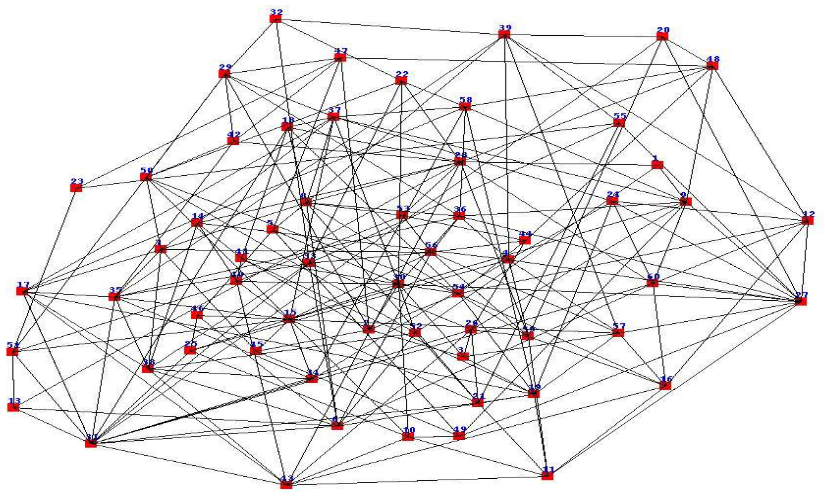
Simulated annealing **Fitness = 1.002**	
Tabu search **Fitness = 1.069**	
Path Relinking + Tabu search **Fitness = 0.600**	

Figure B.1.4 Sample layouts from group 4C in Table 7.1

B.2 Sample Layouts from Datasets of Table 7.2

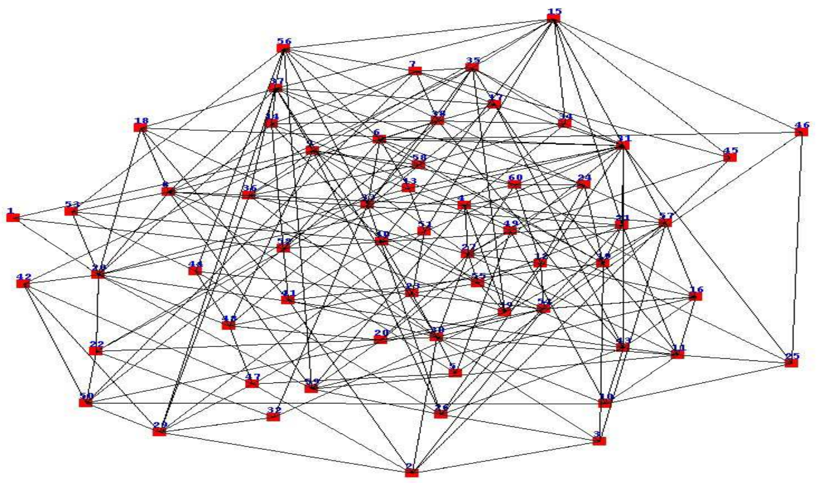
Simulated annealing

Fitness = 0.348



Tabu search

Fitness = 0.385



Path Relinking +
Tabu search

Fitness = 0.335

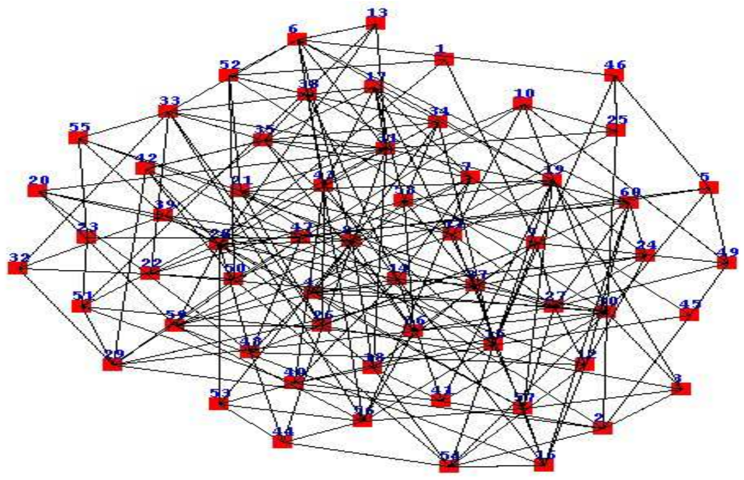
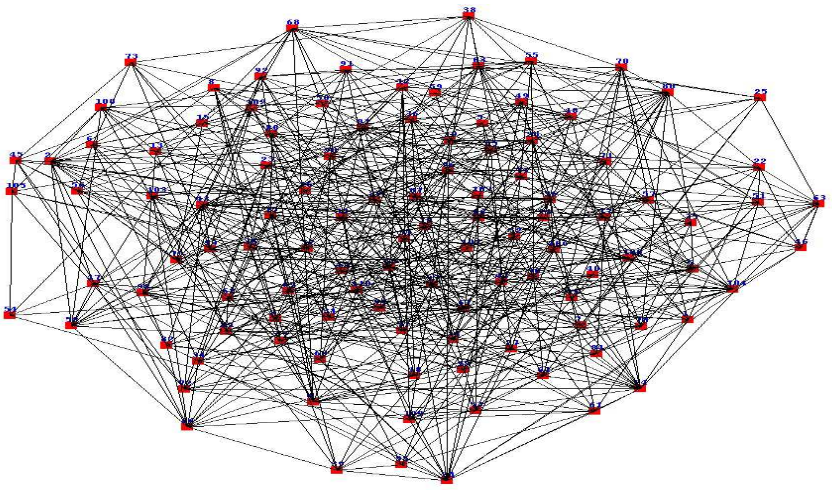


Figure B.2.1 Sample layouts from group 1D in Table 7.2

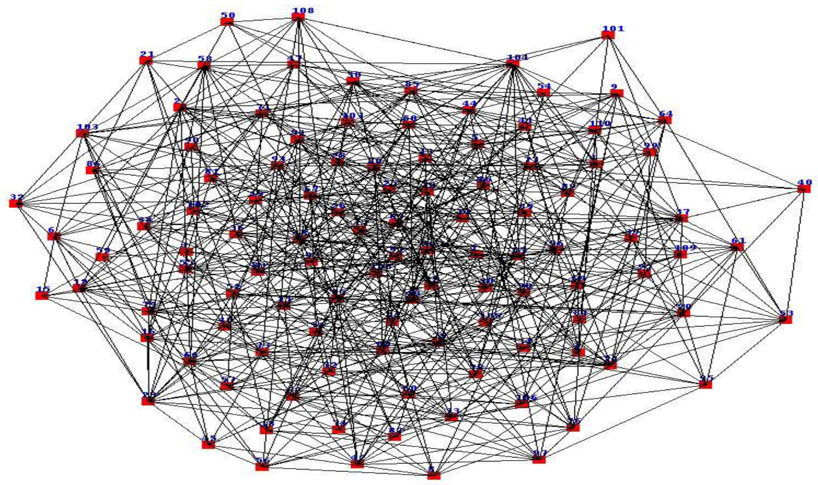
Simulated annealing

Fitness = 0.600



Tabu search

Fitness = 0.625



Path Relinking +
Tabu search

Fitness = 0.301

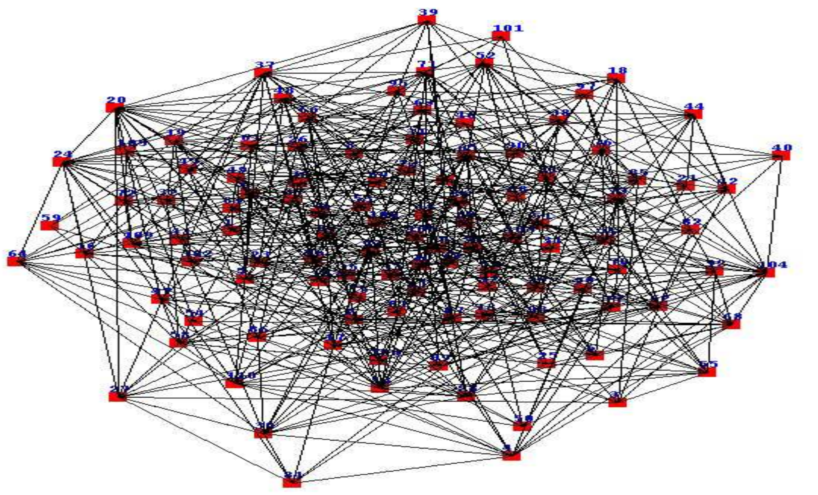


Figure B.2.2 Sample layouts from group 2D in Table 7.2

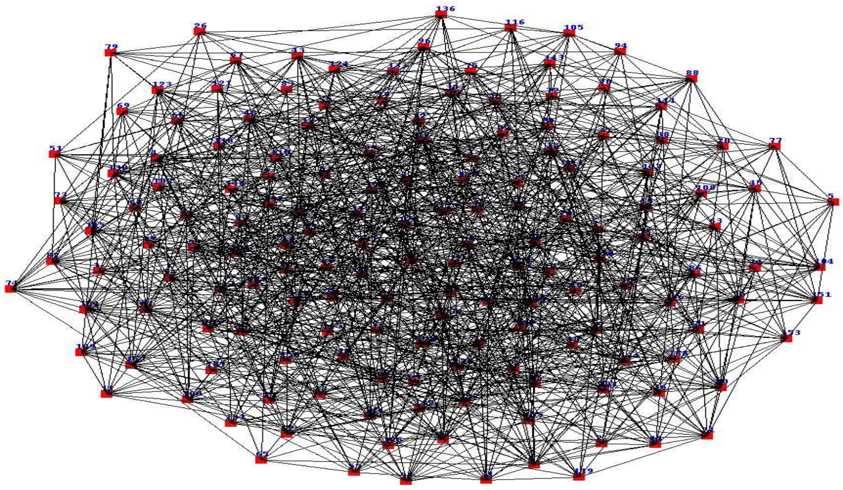
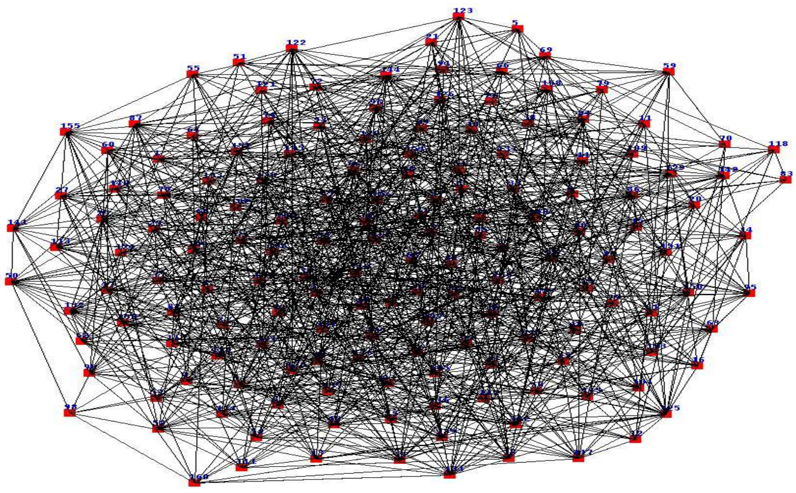
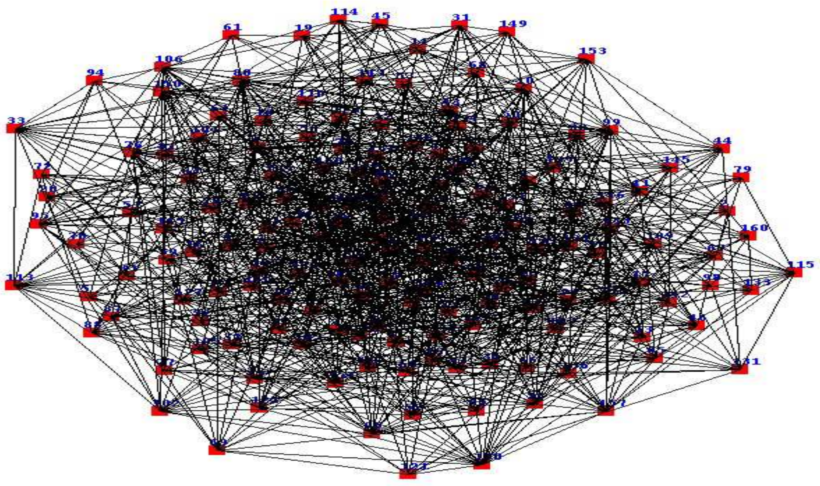
Simulated annealing **Fitness = 0.792**	
Tabu search **Fitness = 0.846**	
Path Relinking + **Tabu search** **Fitness = 0.381**	

Figure B.2.3 Sample layouts from group 3D in Table 7.2

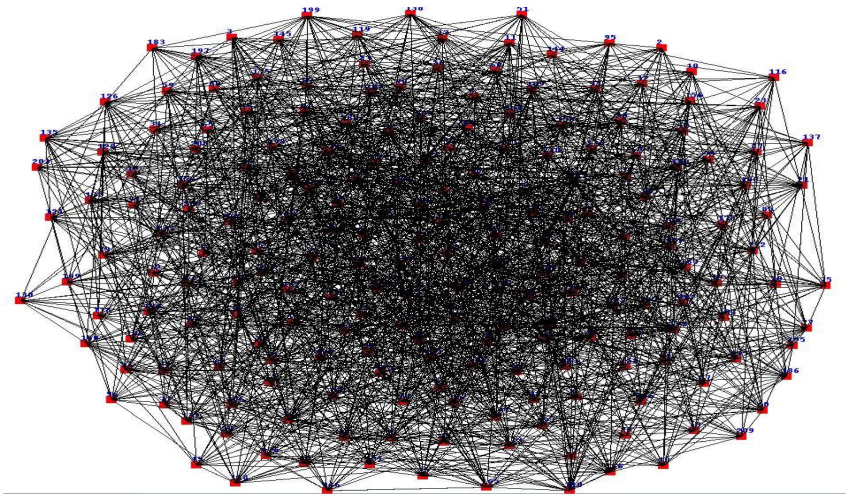
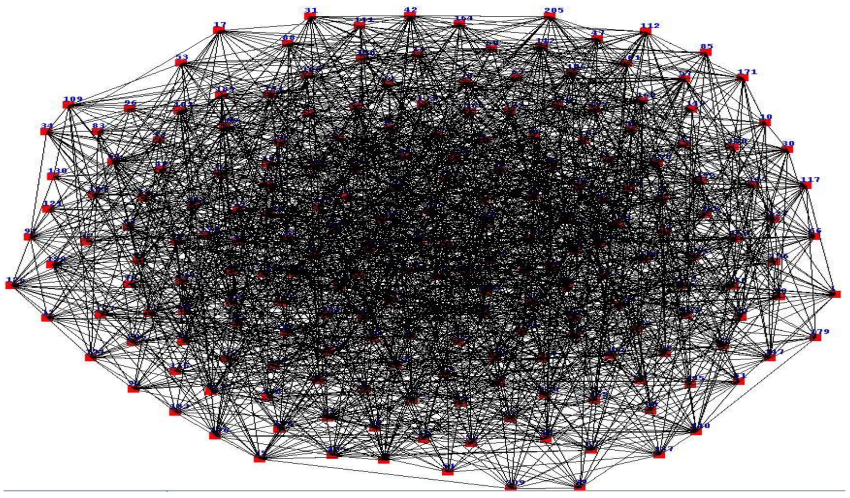
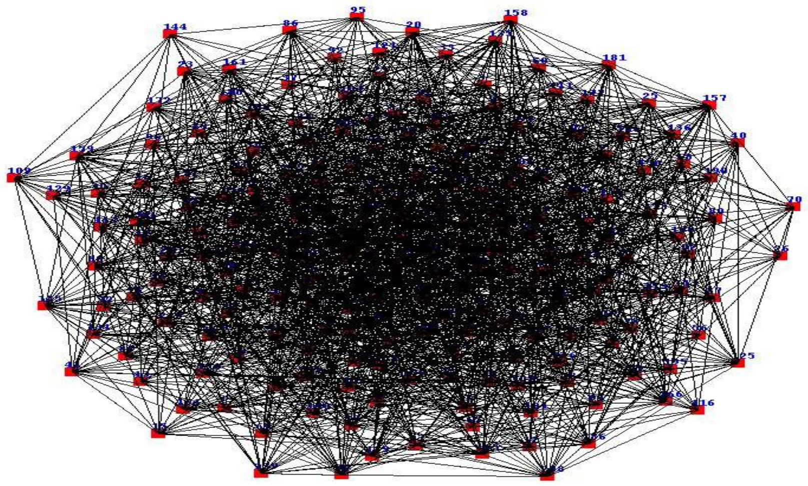
Simulated annealing **Fitness = 0.986**	
Tabu search **Fitness = 1.080**	
Path Relinking + Tabu search **Fitness = 0.431**	

Figure B.2.4 Sample layouts from group 4D in Table 7.2