Department of Mathematics and Statistics

Graph Theoretic Based Heuristics for the Facility Layout Design Problem

Yaya S Kusumah

This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University of Technology

May 2001

DEDICATION

For my wife Rita Rosalina, my daughters Astri Yulianti
and Ratih Prabandari, and my beloved son Purnama Aji Kusumah,
who shared my vision and encouraged me to pursue it.

CERTIFICATION

I certify that the work presented in this thesis is my own work and that all references are duly acknowledged. This thesis has not been submitted previously, in whole or in part, in respect of any academic award at Curtin University of Technology or elsewhere.

Yaya S. Kusumah

May 2001

SUMMARY

The facility layout design problem is concerned with determining the arrangement and configuration of facilities, which optimizes a prescribed objective such as profit, cost, or distance, and which satisfies various prescribed constraints pertaining to available resources. In industry, facility layout design problems arise in manufacturing, in warehousing, and in various assignment type situations. The solution of these problems impacts on the viability of the industry. For example, material-handling costs which can comprise between 30 and 75% of the total manufacturing costs, can be reduced by using the optimization methods associated with the facility layout design. In the service industries, facility layout design problems arise in the location of emergency facilities (such as ambulance, fire stations) and in the allocation of space. The solution of these location problems impacts on the well being of the community.

Mathematically, the facility layout problem has been modelled as: a quadratic assignment problem, a quadratic set covering problem, a linear integer programming problem, a mixed integer programming problem, and a graph theoretic problem. The problem has been shown to be NP-complete. This computational difficulty has led researchers to consider suboptimal solutions generated by heuristic approaches. There are a number of heuristic procedures that have been proposed for solving the facility layout design problem, including Simulated Annealing, Tabu Search, Expert Systems, and Graph Theoretic Algorithms. The most successful heuristic approaches are based on graph theoretic concepts.

In this thesis we focus our study on constructive graph theoretic based heuristics for determining an optimal arrangement and configuration of facilities with the objective of maximizing the total benefit. We are particularly interested in constructive heuristics, which can produce a maximum-weighted planar graph as a final solution. Our contribution is the development, implementation, and testing of three new algorithms. Computational results, based on 4200 randomly (uniform and normal distribution) generated problems, demonstrate the value of our methods. We also present the performance of each algorithm when various initial solutions are applied.

Chapter 1 provides the background of the facility layout design, including the notation, terminology and general concepts as well as a summary of the thesis.

Chapter 2 provides a comprehensive survey of the facility layout design problems. This includes models and methods of solution based on exact algorithms (including the branch and bound method and the cutting plane method), as well as heuristic algorithms. We detail the main constructive graph theoretic based heuristics in the literature: the Deltahedron Method, the Green-Al Hakim Algorithm, the Leung's Constructive Heuristic, the Kim-Kim Algorithm, the Wheel Expansion Method, TESSA and the String Processing Algorithm. We also briefly discuss the non-graph theoretic heuristics including simulated annealing, tabu search, and expert systems.

In Chapter 3 we present three new graph theoretic based heuristics. These heuristics are constructive and the solution is built up, starting with an initial layout of four facilities, by an insertion process. Our algorithms have two important features. Firstly, they allow for previously chosen edges to be removed at each insertion step. Secondly, they do not restrict the type of maximal planar graph produced. Computational results and a comparative analysis of the main graph theoretic based heuristics are provided. The analysis is based on 4200 randomly generated test problems (from uniform and normal distribution). The test problems consist of 30 data sets with the number of facilities ranging from 5 to 100 in increments of 5.

Chapter 4 is devoted to the performance of graph theoretic based heuristics when different types of initial solutions are applied. Examples show that the final

solution is sensitive to the initial solution. Computational results indicate that for most algorithms, the best type of initial solution is the selection of four facilities which yield the best objective function value contribution. However, this does not always coincide with that proposed in the original description of the algorithms.

We conclude this thesis by discussing some future research that can be carried out on the facility layout design problem, particularly in graph theoretic based heuristics.

ACKNOWLEDGEMENTS

It is with great pleasure and gratitude that I acknowledge the many people whose expertise and encouragement contributed immensely to the creation of this thesis. A project of such magnitude could not have been completed without the help of many talented and committed people.

First and foremost, I would like to express gratitude for the continued help, constant encouragement, suggestion and full support of my supervisor Prof. Dr. Louis Caccetta, The Head of Department of Mathematics and Statistics, Curtin University of Technology. His feedback and suggestion have certainly improved my thinking on how research should be conducted and how the structure of the thesis should be presented.

I truly appreciate the help given by all the staff in the Department of Mathematics and Statistics, Curtin University of Technology, particularly for all the facilities I used during my study.

A special word of thanks goes to all officers of the PGSM Project (the Indonesian Secondary School Teachers Development Project), particularly Drs. Benny Karyadi M.A., Dr. Simbolon, Dra. Siti Faizah, and Anita, for their help, support, and valuable assistance during my Ph.D. studies.

My deep appreciation also due to the Rector of the Indonesian University of Education, Prof. Dr. H.M. Fakry Gaffar, M.Ed., and the Deputy Rector for Academic Affairs, Prof. Dr. Said Hamid Hasan, M.A., for their support and encouragement during my study in Western Australia.

I wish to express my thanks to the dean of the Faculty of Mathematics and Science Education, the Indonesian University of Education, Drs. Harry Firman,

M.Pd., and the Head of Department of Mathematics Education, the Indonesian University of Education, Drs. Didi Suryadi, M.Ed., who gave me invaluable contribution and encouragement during this entire study.

In any group of people, there are also those that distinguish themselves by going above and beyond. I have been extremely fortunate to have a number of seniors: Dr. R.K. Sembiring of Bandung Institute of Technology, Prof. Dr. Ahmad Hinduan, M.Sc., Prof. Dr. Utari Sumarmo and Drs. Kosim Rukmana, M.Si. of the Indonesian University of Education. They always give me strong motivation to continue my study further to Ph.D. program overseas.

My research effort was assisted by John E. Middle of the Department of Manufacturing Engineering, Loughborough University, United Kingdom, and Drs. Turmudi, M.A. of the Indonesian University of Education, who provided me with some important journal papers. I gratefully acknowledge this invaluable help.

My heartfelt thanks to my wife, Rita Rosalina, for all her sacrifice and the incredible amount of patience during my study in Perth, Western Australia. Sincere thanks also due to my daughters, Astri Yulianti and Ratih Prabandari, and my beloved son Purnama Aji Kusumah, who have been waiting for their father for a long period of time.

Last but not least, I am most fortunate to have helpful and friendly colleagues in the Department of Mathematics and Statistics, Curtin University of Technology, Western Australia; particularly all the postgraduate students who shared ideas, humor, opinions, as well as happiness and experiences.

Perth, Western Australia

May 2001

CONTENTS

CHAPTER 1
INTRODUCTION

Typically facility layout design problems involve the determination of an optimal arrangement and configuration of facilities. For our purposes, a facility is any physical entity that assists with the production of a product or the provision of a service. Some examples of facilities include: machine tool, workstation, manufacturing cell, building, airport, medical centre, etc. The objective may be: to minimize a quantity such as cost, space, material movement or distance travelled; or to maximize a quantity such as output, traffic flow, or benefit. Usually, the problems are further complicated by the need to satisfy a number of constraints that relate to resource availability, safety, sequence, time, or shape.

Facility layout design problems have been studied for some time and considerable attention has been given to this subject over the past few years. Researchers from many disciplines, including: operations researchers, architects, management scientists, engineers, economists, urban planners, and regional scientists, have contributed to the development of the area. Their combined effort has resulted in methods to solve problems arising in material handling, physical arrangement of facilities, cost minimization, and material movement.

In industry, facility layout design problems arise in manufacturing (layout of machines and workstations); warehousing (layout, material handling procedures); and various assignment type situations (aircrafts to gates, files to cylinders of a disk). In the service industries, facility layout design problems arise in the location of emergency facilities (ambulances, fire stations, police stations) and in the

allocation of space (office buildings, shopping centres, airports, etc). In solving management problems, the facility layout problem techniques have been used in hospital organizations to improve the patient's perceived environment.

The solution of facility layout problems impacts on the viability of an industry. For example, material-handling costs which can comprise between 30 and 75% of the total manufacturing costs can be reduced by using the optimization methods associated with the facility layout design. Further, any savings in material handling obtained through a better physical arrangement of the work centre, departments or production cells, is a direct contribution to the efficiency of operation.

Facility layout involves long-term planning. Modifications or rearrangements of an existing facility layout represents a large expense and can often not be accomplished easily, not only in terms of location, but also in processing time. As changing a facility layout is time-consuming and very expensive, altering a design decision has to be implemented only after a careful evaluation study is completed. This means we have to avoid frequent changes to the layout, as the associated fixed costs would burden the profitability of the organization. It is important to design new facilities carefully and to use established facilities efficiently.

Modern flexible manufacturing systems and service industries have experienced dramatic developments over recent years. In manufacturing, the layout of flexible machines has cost implications for the maintenance and the operation of the material handling system, load balancing, production delays, work-in-process, and for many other factors. As an illustration, the cost of material flow depends not only on the quantity of material exchanged between two machines, but also on the relative location of these machines. In service industries, a number of complex issues arise in the management of the service system. For example, designing and planning the arrangement in a hierarchy, which consists of determining the services to be provided, manufacturing or service to be used, as well as the number and types of manufacturing or service equipment required.

The methods used to solve the facility layout design problems are also useful in tackling the facility location problems. For example, the location of emergency service facilities, including ambulance and fire stations, have to be determined properly, so that the community needs can be adequately covered. The optimal location of these facilities in a region, which impacts significantly on the safety of the community, is a major concern and challenge for modern city planners. Considerable effort has gone into the optimal location of service emergencies in some major cities in the world. For example, in Canada in efforts to improve ambulance coverage in modern cities, a heuristic method was implemented using a graph model of the network (Gendreau *et al.*, 1988). The allocation of emergency vehicles in Louisville, Kentucky, used an optimization model which decreased the response time by 36% (Repede and Bernado, 1994). Some efficient models also have been implemented in Tucson Arizona, in reassessing the ambulance deployment (Goldberg *et al.*, 1990); and in Bangkok, in reducing the number of ambulance in the city without decreasing the level of service (Fujiwara *et al.*, 1987).

As described by Kusiak and Heragu (1987), the facility layout problem has been modelled as: a quadratic assignment problem, a quadratic set covering problem, a linear integer programming problem, a mixed integer programming problem, and a graph theoretic problem. Sahni and Gonzalez (1976) has shown that the problem is computationally difficult. In fact, it is in the so called NP-complete class of problems. Therefore, even though a number of exact algorithms have been proposed, computational difficulties arise in large applications. These computational difficulties have led researchers to consider suboptimal solutions generated by heuristic approaches.

There are a number of heuristic procedures that have been proposed for solving the facility layout design problem. These approaches include Simulated Annealing (Heragu and Alfa (1992), Kouvelis *et al.* (1992), Souilah (1995), Meller and Bozer (1996)); Expert Systems (Fisher and Nof (1984), Kumara *et al.* (1987, 1988), Leskowsky *et al.* (1987), Kusiak and Heragu (1988), Abdou and Douta (1990), Sirinaovakul and Thajchayapong (1994)); Tabu Search (Glover (1986,

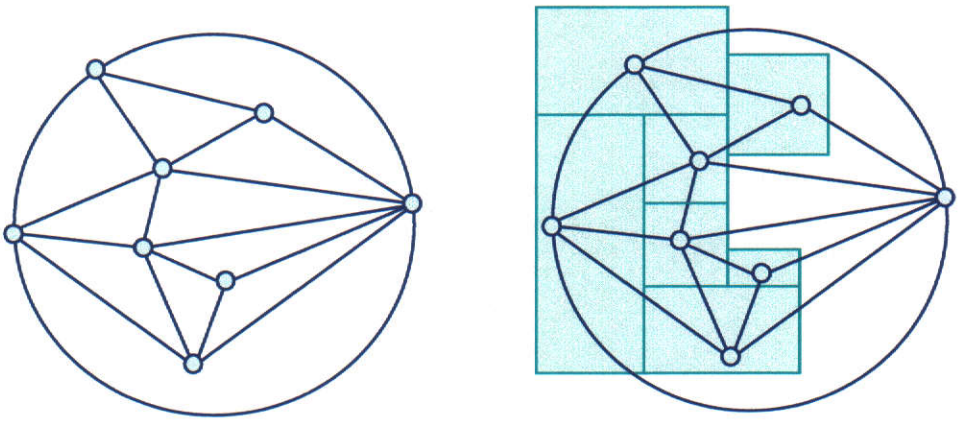
1990), Skorin-Kapov (1990), Taillard (1991), Chiang and Kouvelis (1996), Hertz *et al.*, 1997)); Graph Theoretic Algorithms (Foulds and Robinson (1978), Foulds *et al.* (1985), Green and Al Hakim (1985), Leung (1992), Eades *et al.* (1982), Kim and Kim (1995), Boswell (1992), and Caccetta and Kusumah (1998, 1999, 2000)).

The most successful heuristic approaches are based on graph theoretic concepts. A graph is a simple mathematical structure that consists of a set of vertices some of which are joined by edges. In the graph model of a facility layout problem, the vertices represent facilities and the edges represent adjacencies. It is assumed that the desirability of locating each pair of facilities adjacent to each other is known; this information is represented as an edge weight in the graph model. When the edge weight represents benefit, the problem is to find an arrangement which maximizes the total benefit. Thus, given a weighted graph G, the facility layout problem is to find a maximum weighted spanning subgraph G' of G that is "planar". A planar graph is a graph that can be embedded on the plane so that edges meet only at their ends (there is no crossover). A planar graph G is called maximal planar if adding an edge between any two non-adjacent vertices in G gives rise to a non-planar graph. The planarity requirement comes from the fact that the final solution needs to be a "block plan".

From a maximal planar graph solution obtained, the corresponding block plan can be easily drawn by converting a dual graph to a block layout (Giffin *et al.* (1986), Green and Al-Hakim (1985), Hassan and Hogg (1987, 1989, 1991), Rinsma *et al.* (1990), Al-Hakim (1992), Irvine and Rinsma (1997), and Watson and Giffin (1997)). Figure 1.1 illustrates an example of a block plan (block layout) together with the corresponding maximal planar graph.

The graph theory approach to the facility layout design has some advantages. An important contribution is that it provides an easy bound on the objective function, obtained from the summation of the $3n$-6 best (in terms of weight) edges among the possible $n(n\text{-}1)/2$ edges, where n is the total number of vertices. Note that a maximal planar graph on n vertices must have $3n$-6 edges. This bound has been used to assess the performance of heuristics. An important

computational advantage of the graph theory approach is that it does provide a good quality solution without the use of planarity testing.



(a) Maximal planar graph. (b) Block layout.

Figure 1.1. A graph and its corresponding block layout.

In this thesis we focus our study on graph theoretic based heuristics for determining an optimal arrangement and configuration of facilities with the objective of maximizing the total benefit. We are particularly interested in constructive heuristics, which can produce a maximum-weighted spanning subgraph G' of G (that is planar) as a final solution. A number of new heuristics are developed and their performance in terms of solution quality and solution time are presented.

Before we present our work we shall first introduce some basic notation and terminology, which will be used throughout this thesis. This presentation will be given in Section 1.1, followed by a brief review and summary of the thesis in Section 1.2.

1.1 Notation and Terminology

As there is a wide variation in graph theory notation and terminology used in the literature, we discuss, in this section, the basic notation and terminology that will be used in discussing the facility layout design problem throughout this thesis. Some

of our notation and terminology follows that of Bondy and Murty (1976), and Chartrand and Lesniak (1986).

We begin with some basic graph theoretic concepts. A **graph** G is an ordered triple $(V(G), E(G), \psi(G))$ consisting of a non-empty set $V(G)$ of **vertices** and a set $E(G)$, disjoint from $V(G)$, of **edges**, and an **incidence function** ψ_G that associates with each edge of G an ordered pair of (not necessarily distinct) vertices of G. If u and v are vertices of a graph G identified with an edge e, that is $\psi_G(e) = (u,v)$, then e is said to **join** u and v, and we write $e = (u,v)$. The vertices u and v are called the **ends** of e. We also say that the vertices u and v are **incident** with the edge e, and vice versa. Further, we also say that u and v are **adjacent**.

We say that a graph G is **finite** if both $V(G)$ and $E(G)$ are finite sets. The **order** of a graph G, denoted by $\nu(G)$, is the number of vertices of G. The **degree** of a vertex of a graph G, denoted by $\deg_G(v)$, is the number of edges of G incident with u. In any graph, the sum of all vertex degrees is equal to twice the number of edges. An edge of G is called a **loop** if it joins a vertex to itself. Two or more edges joining the same pair of vertices are called **multiple** edges and a graph with no loops and no multiple edges is called a **simple** graph.

If G is a graph such that each vertex in G is adjacent to all other vertices, then G is called a **complete** graph. If G is a complete graph on n vertices, then G is denoted by K_n. Note that K_n has $n(n\text{-}1)/2$ edges.

A graph G is **bipartite** if $V(G)$ can be partitioned into two subsets X and Y such that each edge joins a vertex of X to a vertex of Y; X and Y are called **bipartitioning sets** of G. A **complete bipartite** graph is a bipartite graph with bipartitioning sets X and Y in which each vertex of X is joined to each vertex of Y; such a graph is denoted by $K_{m,n}$, when $m = |X|$ and $n = |Y|$.

Graph G is said to be **isomorphic** to graph H, denoted by $G \cong H$, if there exists a one to one correspondence between their vertex sets that preserves adjacency.

Let G be a graph without loops, with n vertices labelled $v_1, v_2, v_3, \ldots, v_n$. The **adjacency matrix** $A(G)$ is the n x n matrix in which the entry in row i and

column j is the number of edges joining the vertices v_i and v_j. For a simple graph G an adjacency matrix $A(G)=[a_{ij}]$ can be defined by

$$a_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E(G) \\ 0, & \text{if } (i,j) \notin E(G). \end{cases}$$

A **walk** in graph G is a finite non-empty alternating sequence $W = v_0e_1v_1e_2v_2 \ldots e_kv_k$, where $1 \le i \le k$, and the ends of e_i is v_{i-1} and v_i. In this sequence, v_0 is called the **origin** of W, v_k is called the **terminus**, and the other vertices are called **internal** vertices of W. W is said to be a walk from v_0 to v_k and denoted by (v_0,v_k)-walk. If the vertices $v_0, v_1, \ldots, v_k$ are distinct, then W is called a **path**. A walk is **closed** if it has positive length and its origin and terminus are the same. A closed walk whose origin and internal vertices are distinct is a **cycle**. A cycle of length k, where k is the number of edges, is denoted by C_k, and is called k-cycle.

A graph H is called a **subgraph** of graph G if the set $V(H)$ is a subset of $V(G)$, the set $E(H)$ is a subset of $E(G)$, and ψ_H is the restriction of ψ_G to $E(H)$. We say that H is a **spanning subgraph** of G if H is a subgraph of G and $V(H) = V(G)$.

Let V' be a non-empty subset of $V(G)$. The subgraph of G **induced** by V', denoted by $G[V']$, is a graph with vertex set V' and $E(G[V']) = \{(u,v) \in E(G) : u,v \in V'\}$. Further, $G-V'$ is the subgraph obtained from G by deleting the vertices of V' together with their incident edges. Similarly, for E' a non-empty subset of $E(G)$, the edge-induced subgraph of G, denoted by $G[E']$, is a graph with edge set E' and $V(G[E']) = \{u \in V(G) : u$ is an end vertex of an edge e of $E'\}$.

A graph G is called a **weighted** graph if each edge e in G is assigned a positive real number, called the **weight** of e, and denoted by $w(e)$. The weight of a subgraph H of G, denoted by $w(H)$, is the sum of the weights

$$w(e_1)+w(e_2)+w(e_3)+ \ldots + w(e_t),$$

of the edges of H.

Vertices u and v of graph G are said to be **connected** if there exists a (u,v)-path in G. A graph G is **connected** if every two of its vertices are connected. A

graph that is not connected is **disconnected**. A **tree** is a connected graph which contains no cycles. A **spanning tree** of G is a spanning subgraph of G that is a tree.

A point of intersection in a graph is called a **crossover**. If G is a graph, which is drawn on a surface S such that no two edges crossover, then we say G is **embedded** on S. A graph that can be embedded on the plane without crossover is called a **planar** graph. If a planar graph is already embedded on the plane so that it has no crossover, it is called **a plane** graph. If G is a planar graph, then any plane drawing of G divides the plane into regions, called **faces**. One of these faces is **unbounded**, and is called the **infinite** face. The **degree of face** f is the number of edges encountered in a walk around the boundary of the face f. A **plane triangulation** is a plane graph in which each face has degree three. If G is a plane triangulation and n is the number of vertices in G, then the total number of edges and faces in G are $3n$-6 and $2n$-4 respectively. The graphs in Figure 1.2 give some illustrations of plane triangulations.

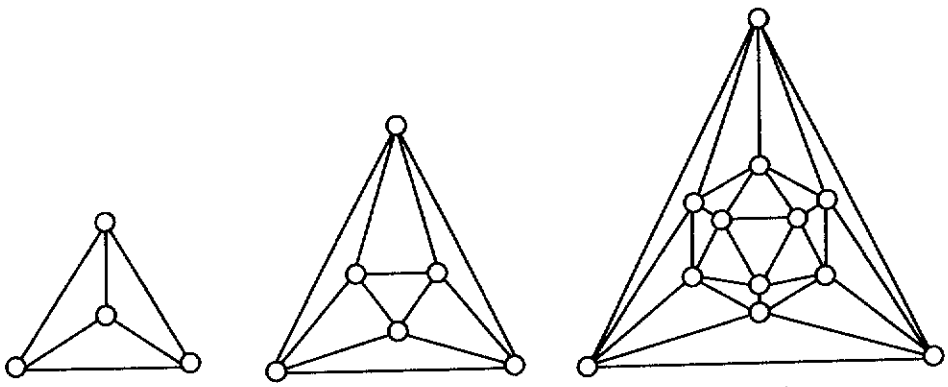


Figure 1.2. Plane Triangulations

The sum of all degrees of the faces is equal to twice the number of edges in the graph, since each edge either borders two different faces or occurs twice when we walk around a single face. This result can be regarded as a sort of "handshaking lemma" for the faces of a planar graph, and we shall refer it as the handshaking lemma for planar graphs.

A planar graph G is called **maximal planar** if adding an edge between any two non-adjacent vertices in G gives rise to a non-planar graph. Any embedding of a maximal planar graph G with order $n \geq 3$ can be classified as a **plane triangulation** as each region or face is bounded by three edges. A maximal planar graph is also called **a triangulated planar graph**, and triangulated plane graphs are referred to as **triangulations**. Some properties of maximal planar graphs have a contribution to the finding of maximum-weight maximal planar graph. A graph G is maximally planar if and only if G is planar triangulation.

Theorem 1.2.1 (Euler's Formula)

Let G be a connected planar graph, and let n, m, and f denote the numbers of vertices, edges, and faces in a plane drawing of G. Then $n-m+f=2$.

Theorem 1.2.2

Let G be a maximal planar graph, with n vertices and m edges. Then $m = 3n - 6$.

Proof: Let f be the number of regions in graph G. The boundary of each region in G is a triangle (a face with degree 3), and each edge is on the boundary of two regions. The sum of edges on the boundary of a region over all regions is $3f$. This sum counts each edge twice, so that $3f = 2m$. By using Theorem 1.2.1, we have $m = 3n - 6$.

Corollary 1.2.3

Let G be a planar graph, with n vertices and m edges. Then $m \leq 3n - 6$.

Corollary 1.2.4

Let G be a maximal planar graph, with n vertices and f faces. Then $f = 2n - 4$.

Proof: Since G is a maximal planar graph, then according to Theorem 1.2.1 we have $f = 2 + m - n$, and from Theorem 1.2.2, we have $m = 3n - 6$. Therefore, $f = 2n - 4$.

Corollary 1.2.5

Every planar graph contains at least one vertex of degree at most 5.

Proof: To prove this corollary, we use an indirect proof. Suppose to the contrary that G is planar, with each vertex having degree greater than 5. Then

$$2|E| = \sum_{v \in V} \deg v \geq 6|V|,$$

which gives $|E| \geq 3|V|$, a contradiction.

The **dual** graph G' of a planar graph G is a graph which is obtained by placing a vertex in each face of G, including the infinite face and joining every pair of vertices whose faces have a common edge. A dual graph G' of a planar graph G is also planar.

It is obvious that the number of edges in G and G' are equal. However, the number of faces and vertices are interchangeable. Figure 1.3 displays a planar graph (solid lines) and its dual (dotted lines).

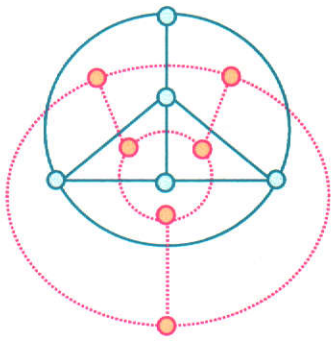


Figure 1.3. A graph (solid lines) and its dual (dotted lines).

To understand the structure of planar graphs we need to study two graphs, which are important in discussing planar graphs: K_5 and $K_{3,3}$.

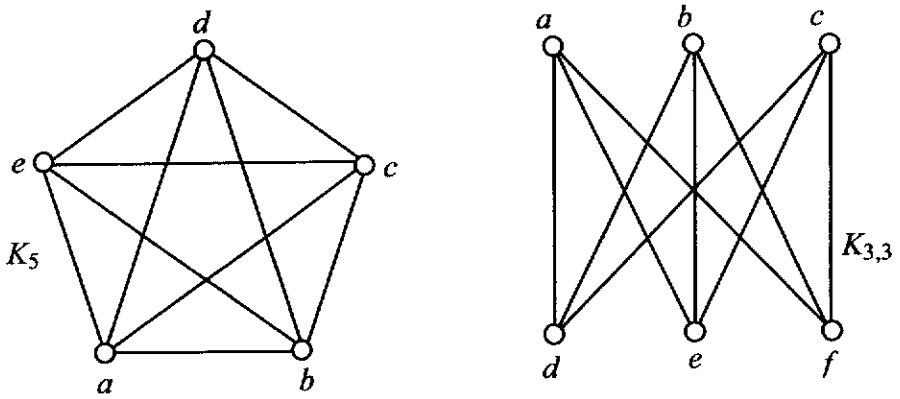


Figure 1.4. Non planar graphs K_5 and $K_{3,3}$.

Theorem 1.2.6

The graphs K_5 and $K_{3,3}$ are non-planar.

Proof: To show that K_5 is non-planar we use a contradiction. Suppose that this graph is planar. According to Corollary 1.2.3, the relation between its vertices and edges has to follow $m \leq 3n - 6$. However, we know that this graph has 5 vertices and 10 edges, which gives $10 \leq 3(5) - 6 = 9$, a contradiction. Therefore K_5 is non-planar.

To see that $K_{3,3}$ is a non-planar graph, suppose that this graph is planar. This graph is complete bipartite and in any drawing it always has a cycle of 6 vertices (a cycle of length 6). Suppose, without loss of generality, we take the cycle *adbecfa* (see Figure 1.5). Now we have to add the other edges to this cycle.

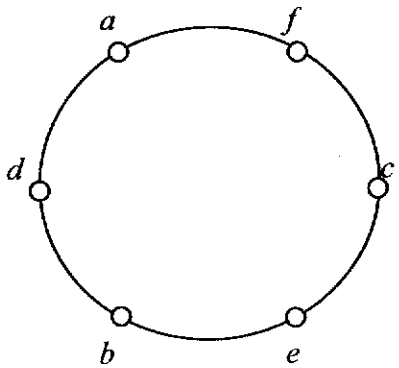


Figure 1.5. A cycle of 6 vertices.

Without loss of generality, we add edges incident to vertices a and d: (a,e) and (c,d), as displayed in Figure 1.6. Edge (b,f) is one of the edges that has to be inserted. However the insertion of this edge results in a crossover with edge (a,e) in the inner side, or edge (c,d) in the outer side of the graph. This means that graph $K_{3,3}$ is non planar.

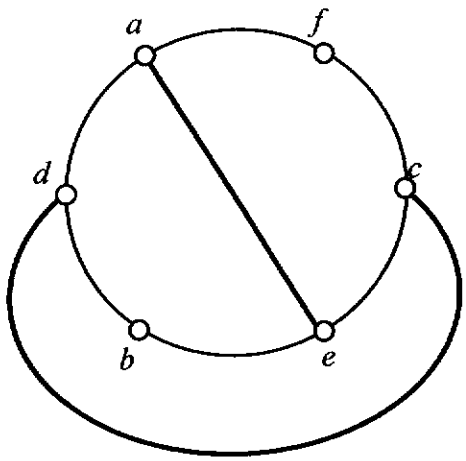
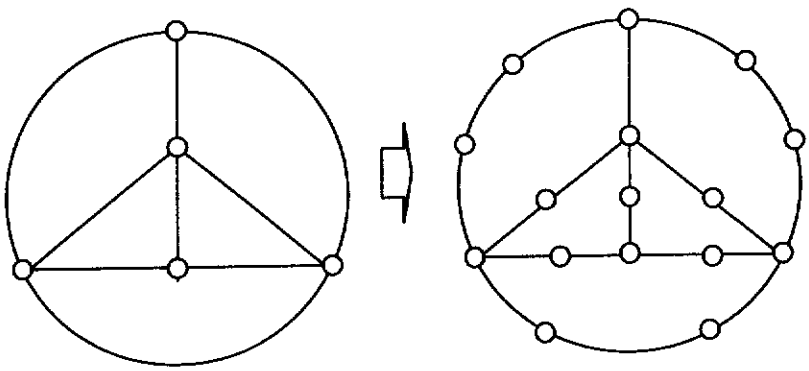


Figure 1.6. Adding 2 edges to the cycle.

The restriction on the number of edges in a planar graph given in Corollary 1.2.3 is often useful for showing that a graph is non-planar. We used this to prove that K_5 is non-planar. However, this method does not work the other way around, as there are so many graphs (including $K_{3,3}$), which satisfy this inequality but they are not planar. It is evident from simple observations that a graph, which contains a non-planar graph as a subgraph, is also non-planar. This means that any graph containing K_5 or $K_{3,3}$ is non-planar.

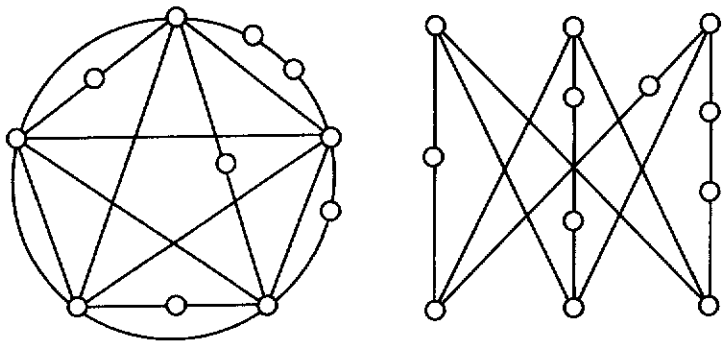
Suppose we have an insertion of vertices of degree 2 into edges of a graph G, as shown in Figure 1.7 below. Graph H is called a **subdivision** of graph G.



Graph G Graph H

Figure 1.7. Inserting vertices of degree 2 into the edges of a graph.

The insertion of a vertex of degree 2 cannot affect the planarity or non-planarity of a graph. Therefore, every subdivision of a graph preserves the planarity property. For example, graphs P and Q in Figure 1.8 are non-planar since they contain a subdivision of K_5 and $K_{3,3}$, respectively.



Graph P Graph Q

Figure 1.8. Graphs which contain a K_5 or a $K_{3,3}$ as a subdivision.

Perhaps all non-planar graphs contain as a subgraph a subdivision of K_5 or $K_{3,3}$. This is in fact true and stated formally as:

Theorem 1.2.7 (Kuratowski's Theorem)

A graph is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$ as a subgraph.

Constructive proofs of this important result do exist and lead to efficient algorithms for planarity testing. However, any algorithm, which applies a planarity testing routine in solving the facility layout design problem, is not time efficient (Foulds and Robinson, 1978).

Now we discuss the concept of braced edges. In a plane triangulation, every edge is a common edge between two incident triangular faces. Suppose we have a graph containing two faces (a,b,c) and (a,b,d), where (a,b) is the common edge. In this graph (a,b) is **braced** if (c,d) exists in the graph and it braces (a,b). If (c,d) does not exist, (a,b) is **unbraced**.

A single edge can brace more than one edge of a planar graph. An example is illustrated in Figure 1.9. In this graph, the edges (a,b), (a,d), (a,e), (b,c), (c,d), and (c,e) are braced (in red colour), whereas (b,d), (b,e), and (d,e) are unbraced (in blue colour). It can be seen that (b,e) braces (a,d), and (c,d); (b,d) braces edges (a,e), and (c,e); and (d,e) braces (a,b), and (b,c).

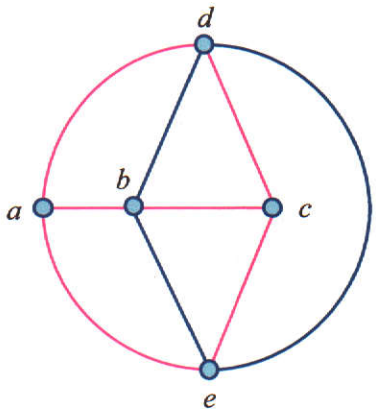


Figure 1.9. A planar graph with braced edges.

There are also some graphs having no braced edges. Graphs P and Q in Figure 1.10 illustrate triangulated graphs without braced edges.

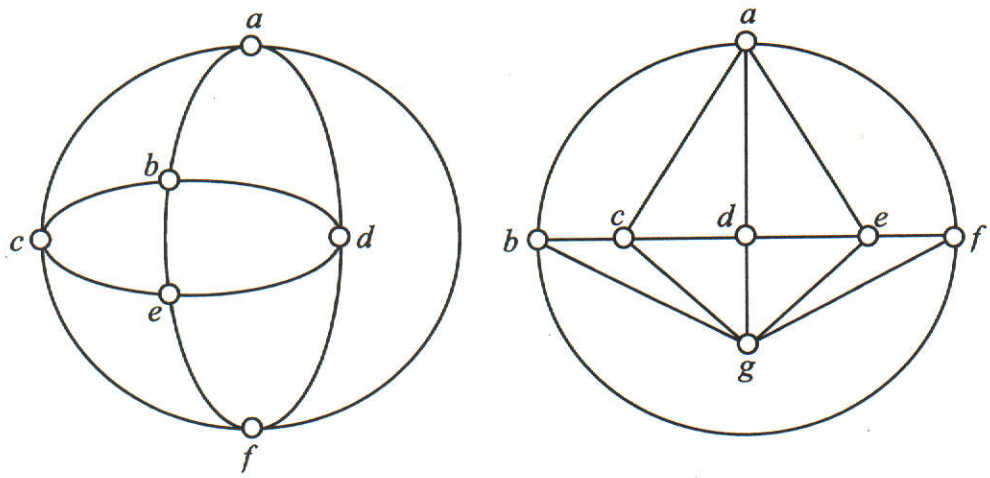


Figure 1.10. Triangulated graphs without braced edges.

The plane triangulation of order 3, which corresponds to a plane embedding of K_3, has two faces: inner faces and an outer face. This plane triangulation has the same boundary, consisting of just 3 vertices; hence the concept of braced and unbraced edges cannot be applied to this triangulation. K_4 is a good example for illustrating braced edges. This graph is the only plane triangulation of order 4, having all edges braced. By inserting a vertex into a face of a plane triangulation of order 4 we can obtain a plane triangulation of order 5. The graph illustrated in Figure 1.9 is a plane triangulation of order 5. This graph is the unique triangulation of order 5. It has 6 edges which are braced, and 3 edges which are unbraced. A plane triangulation of order $n \geq 5$ has at least one unbraced edge.

Theorem 1.2.8 (Eggleton *et al.* (1990), Eggleton and Al-Hakim (1991))

If T is a plane triangulation of order $n \geq 5$, then each edge is either unbraced or the edge which braces it is unbraced.

Corollary 1.2.9

Every plane triangulation of order $n \geq 5$ has at least one unbraced edge.

A plane triangulation of order 6 consists of 3 types. Their embedding are displayed in Figure 1.11. Note that all edges in the first plane triangulation are unbraced. However, in the second and the third triangulation there are 5 unbraced edges (blue colour).

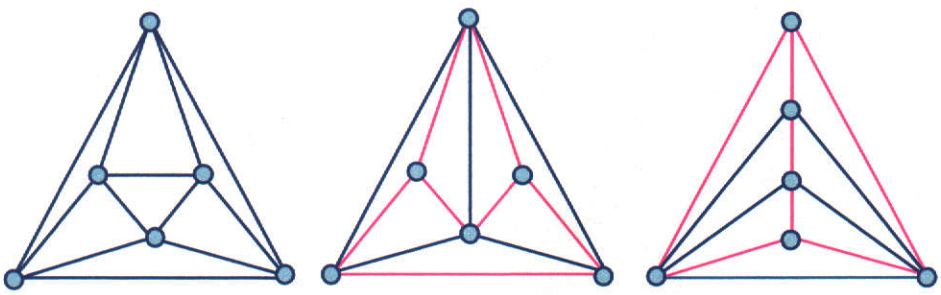


Figure 1.11. Plane triangulations of order 6.

From an existing plane triangulation we can construct a new plane triangulation of the same order. Suppose we have a plane triangulation which contains faces (a,b,c) and (a,b,d), where (a,b) is the common edge, as illustrated in Figure 1.12. If we remove (a,b), followed by inserting (c,d), then we have a new

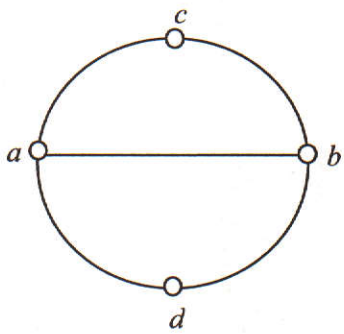


Figure 1.12. A pair of faces with a common edge.

different triangulation (see Figure 1.13). However, we have to ensure that (a,b) is not braced. Otherwise we create multiple edges, and thus the resulting graph is no longer a simple graph.

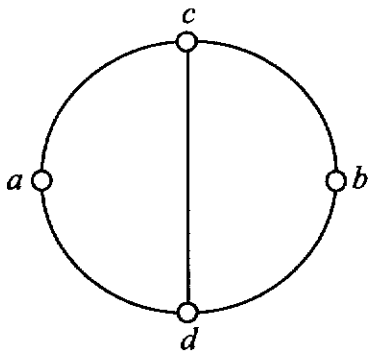


Figure 1.13. A pair of faces with (*c*,*d*) as the common edge.

From a plane triangulation of order n we can construct a plane triangulation of order n+1. Suppose we have a plane triangulation which contains a face with vertices a, b, and c (see Figure 1.14). Inserting vertex x into this face results in 3 new faces and 3 new edges. The old (interior) face is deleted from the plane triangulation. We can see that in the resulting plane triangulation, all three new edges are braced.

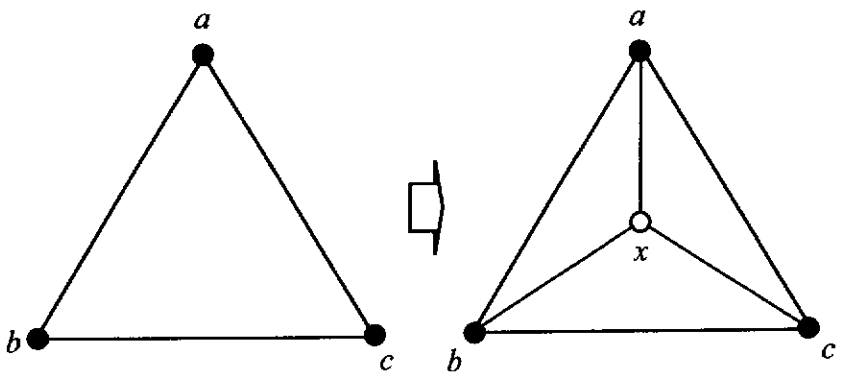


Figure 1.14. Inserting a vertex of degree 3 into a plane triangulation.

From an existing plane triangulation, we can also construct a new higher order plane triangulation by using another procedure. Suppose in a plane triangulation we have a pair of faces (*a*,*b*,*c*) and (*a*,*b*,*d*), where (*a*,*b*) is the common edge as illustrated in Figure 1.15. If we remove (*a*,*b*) and insert vertex v and edges (*a*,*v*), (*b*,*v*), (*c*,*v*), and (*d*,*v*), we have a new plane triangulation with the order one level higher than the original triangulation. Note that in the resulting triangulation all four new edges are unbraced.

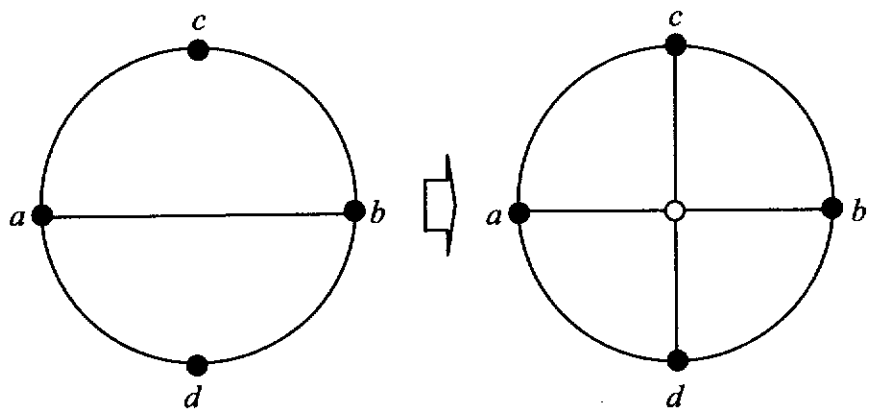


Figure 1.15. Inserting a vertex of degree 4 into a plane triangulation.

It is interesting to see that such conversion can produce a new triangulation without producing multiple edges; so that the resulting graph (subgraph) is also simple and planar. This feature is very important, particularly, when we insert several vertices into a subgraph to obtain a new triangulation of higher order, as will be discussed and investigated throughout this thesis.

Now we discuss the term heuristic and heuristic algorithm. The word "heuristic" is derived from the ancient Greek word "heuriskein", which means to find out or to discover. In English, this word (as an adjective) means serving to indicate or point out, or stimulating interest as a means of furthering investigating.

A heuristic algorithm is a method that attempts to obtain a feasible solution in reasonable time. Whilst there is no guarantee in terms of quality of solution, heuristics have proven extremely valuable in many difficult optimization problems. In the context of facility layout design, a simple (greedy) heuristic can be described in terms of the vertex insertion operation, illustrated in Figure 1.14. The heuristic starts with a triangular face (consisting of three vertices) and inserts the remaining (n-3) vertices (facilities) one at a time, according to the benefit criteria, maintaining the triangulation at all times. The result is a feasible assignment of facilities. This is basically a building or constructive heuristic.

Next we discuss P and NP-complete problems. The theory of computational complexity classifies some problems into two categories "easy" and "hard". All optimization problems, which can be solved by a polynomial-time algorithm, are

classified into the class P of relatively easy optimization problems (those for which efficient algorithms exist). A polynomial-time algorithm solves all instances of a problem using a maximum number of steps that increases polynomially with some measure of the problem size. For many decision problems, all known algorithms have run times that increase exponentially with some measure of the problem size for some instances of the problem. The class NP consists of all problems in P as well as other problems that can be solved by a non-deterministic algorithm in polynomial time. The NP-complete class is a subset of NP having the property that all problems in NP can be reduced in polynomial time to one of them. The Facility Layout Design Problem (FLDP) is NP-complete.

1.2 Review and Summary of Thesis

This thesis focuses on constructive graph theoretic based heuristics for solving the facility layout design problem (FLDP). We consider the facility layout problem where the objective is to maximize benefit. This problem is NP-complete (Sahni and Gonzalez, 1976) and even though a number of exact algorithms have been proposed, computational difficulties arise in large applications. This computational difficulty has led researchers to consider suboptimal solutions generated by heuristic approaches.

We investigate the performance of several algorithms (graph theoretic based heuristics) in the literature, and examine the implementation of various initial solutions to these algorithms. Three new efficient constructive graph theoretic based heuristics are presented. We compare the performance of our heuristics with the heuristics in the literature, based on 4200 randomly (uniform and normal distribution) generated problems.

We provide a survey of the facility layout design problems in Chapter 2. We present the idea of the facility layout design, a number of models for the problem that arise, and several exact and heuristic approaches.

In Section 2.1 we discuss several mathematical programming models for the facility layout design problem, including: a quadratic covering model, a quadratic

set covering model, a linear integer programming model, a mixed integer programming model, and a graph theoretic model. Section 2.2 deals with some established exact algorithms including the branch and bound method and the cutting plane method.

The Branch and Bound (B&B) method has been effectively used to solve a number of computationally difficult problems in the area of combinatorial optimization. The method produces an optimum solution by considering a number of relaxed subproblem. The set of feasible solutions is partitioned into many smaller subsets. At each stage, a promising subproblem is selected and an effort is made to find the best solution for it. If the solution obtained is the best so far, then we update the bound. Otherwise we again partition this subset into two or more smaller subsets (called branching) and the same process is repeated until the best solution is obtained.

The success of the B&B method depends on the branching strategy, the search strategy, and the quality of the bound generated. Depth-first search strategy is often used as a search technique, where a subset selected from the list is explored until an improved feasible solution or a violation of feasibility to the bound is found. Another subset is selected if the solution is infeasible. We then do branching from the parent node in the branch where exploration has not been done. An alternate search in the Branch and Bound method is best-first, in which the subset with the lower bound LB, is chosen to branch from.

Branch and bound algorithms were developed by Gilmore (1962), Lawler (1963), Land (1963), Gavett and Plyter (1966), Pierce and Crowston (1971), Burkard (1973), and Bazaraa (1975). In dealing with the facility layout design, Foulds and Robinson (1976) proposed a branch and bound algorithm, based on a graph theory. Most of B&B methods have an emphasis on solution quality. In Bazaraa and Kirca (1983), and Burkard and Stratman (1978), good quality solutions are obtained, however at considerably higher CPU time.

In Section 2.2 we also discuss the cutting plane (CP) algorithm, which solves integer programs by adding new constraints successively, so that the feasible region of the subproblem is reduced until an integer optimal solution is obtained.

After a cutting plane (or more) is added to the linear program, this program is again solved by using the Simplex method. The CP procedure terminates when no further violation can be found or an optimal solution has been found. If an optimal solution is not obtained, then we subdivide the solution space by embedding the cutting plane procedure in a tree search technique. Again we add new violating inequalities to the set of constraints and solve the associated relaxed problem until an integer optimal solution is found.

The CP algorithm has been used for solving facility layout design problem (Bazaraa and Sherali, 1980) and solving QAPs (Burkard and Bonninger, 1983). It is known that optimal cutting plane algorithms have a very high computing time and storage requirement. As an example, the largest size of the facility layout design problem by using CP algorithm is 8 facilities. Current methods developed for the Travelling Salesman Problem (TSP) could probably be significantly improved, but this is not the focus of this thesis.

In Section 2.3 we deal with various heuristic algorithms, including graph theoretic based heuristics, simulated annealing, tabu search, and expert systems. In this Section we present, in detail, the main graph theoretic based heuristics in the literature, including the Deltahedron Method, the Green and Al-Hakim Algorithm, Leung's Constructive Heuristic, the Kim-Kim Algorithm, the Wheel Expansion Method, TESSA and the String Processing Algorithm. The six aforementioned algorithms are constructive graph theoretic approaches, which employ vertex insertion (except TESSA) in each step to obtain a maximal planar graph. They start by constructing a triangle or a tetrahedron as an initial solution. From a set of 3 or 4 vertices, a triangle or a tetrahedron with the highest weight is constructed. The solution is then grown iteratively by inserting up to 3 vertices into the current partial solution, until all vertices have been inserted. This operation maintains the planarity of the generated graph. The partial subgraphs, as a partial solution in all established heuristics, except TESSA, are always maximal planar.

The String Processing Algorithm, which uses graph theory concepts, manipulates string symbols, corresponding to a graph. It starts by choosing an edge, which has the highest weight, followed by selecting edges, one edge at a time, until

a maximal weighted spanning tree is obtained. By triangulating this tree through edge insertions, we construct a maximal planar graph.

The discussion of graph theoretic heuristics is then followed by the brief description of simulated annealing, tabu search and expert systems. In local search when the solution is trapped in a possibly inferior region, it is often difficult to get out of this region and find any other solution. By using Simulated Annealing, there is a possible move to a better solution. If the new objective function value is not better than that of the previous one, the new solution is accepted based on certain probability criteria.

Tabu Search, which also can avoid the search from being trapped at a local optimal solution, starts from an initial solution. To generate a list of candidate solutions, some local exchange techniques are implemented. If there are many candidate solutions available, a restriction of the search may be used to create a subset of candidate solutions. Then, this subset of solutions is evaluated, followed by a selection of the best solution from the candidate set of solutions. The next best solution is selected if this solution has a forbidden status under a prescribed rule. This selected solution becomes the new current solution. To avoid the search going into an unwanted set of solutions, a tabu list is set up. This can avoid cycling. This procedure is repeated until either the required number of iterations have been performed, or a given CPU time has been reached.

The idea of Tabu Search was initiated in the 1970s, and has been investigated by a number of researchers, including Glover (1986, 1992), Hansen (1986), de Werra and Hertz (1989), Skorin-Kapov (1990, 1991), Taillard (1991), Faigle and Kern (1992), Fox (1993), and Chiang and Kouvelis (1996).

Another method that we discuss in Section 2.3 is that of Expert Systems, which is used to incorporate a qualitative objective function. Thus quantitative and qualitative objectives can be integrated into the model. The series of expert rules is used to evaluate the relativity chart. Expert Systems is devised by Fisher and Nof (1984), Kumara *et al.* (1987, 1988), Leskowsky (1987), Kusiak and Heragu (1988), Abdou and Duta (1990), and Sirinaovakul and Thajchyapong (1994).

Chapter 3 is devoted to the development of three new graph theoretic algorithms. Our algorithms have two important features. Firstly, they allow for previously chosen edges to be removed at each insertion step. Secondly, they do not restrict the type of maximal planar graph produced. The algorithms are described in Section 3.1. Section 3.2 provides computational results and a comparative analysis of the main graph theoretic based heuristics against the new heuristics developed in Section 3.1. The analysis is based on 4200 randomly (uniform and normal distribution) generated problems with n ranging from 5 to 100 in increments of 5. These computational results demonstrate the value of our methods.

Chapter 4 is devoted to the performance of the graph theoretic based heuristics when different initial solutions are used. In Section 4.1 the initial solution types used by each algorithm in the literature are discussed. In this section we also present an example of the sensitivity of certain algorithms to different initial solutions having the same weight. The initial solution types used in the established literature are the best K_3, the best K_4, and K_4. Most of the graph heuristics are very sensitive to the initial solution, and two initial solutions with the same weight, can produce different weight final solutions. Section 4.2 presents the idea of generating initial solutions, particularly on constructing an initial subgraph (the best) K_4 and the best K_3. An initial solution is constructed by selecting 4 vertices (or 3 vertices) in various different ways. They may be selected randomly, or based on the edge weights. In Section 4.3 computational results are displayed. It is found that for most algorithms, the best type of initial solution is the best K_4. The experiment we carried out indicates that the best initial solution on the test problem does not always coincide with that proposed in the original description of the algorithms.

We conclude this thesis, in Chapter 5, by suggesting some computational work which can be conducted in any future research.

CHAPTER 2

A SURVEY OF THE FACILITY LAYOUT DESIGN PROBLEM

As mentioned in the introduction, the facility layout design problem (FLDP) is concerned with the determination of the most effective arrangement and configuration of physical facilities that assists with the production of a product or the provision of a service. Some examples of a physical entity are: machine tool, workstation, manufacturing cell, building, input, instrumentation panel, police stations, civic complexes, sport centres, and medical centres. Usually, the objective is: to minimize a quantity such as production cost, space, material movement or distance traveled; or to maximize a quantity such as output, traffic flows, or profit of production. Often, the problems are further complicated by the need to satisfy a number of constraints that relate to resource availability, safety, sequence, time, and shape. The optimization problems that arise are computationally difficult (NP-complete). Interest in these problems has been maintained over the past forty years or so.

In this chapter we review the various mathematical models and solution methods that have been proposed for the facility layout design problem. Both exact and heuristic solution approaches are discussed. We also present in detail several graph theoretic based heuristics, together with their computational results. Models are presented in Section 2.1 and solution methods are presented in Section 2.3 (Exact) and Section 2.4 (Heuristic).

2.1 Models for the Facility Layout Design Problem

As described by Kusiak and Heragu (1987), the facility layout problem has been modeled as: a quadratic assignment problem; a quadratic set covering problem; a linear integer programming problem; a mixed integer programming problem; a graph theoretic problem. We briefly outline each of these models below.

(i) Quadratic Assignment Problem

Koopmans and Beckman (1957) modeled the facility layout design problem (FLDP) as a quadratic assignment problem (QAP). In this model the objective function is a quadratic function and the constraints are linear functions of the variables. More precisely, the model can be described as follows.

Let n : total number of plants or locations,

a_{ij} : the fixed cost of locating and operating plant i at location j,

f_{ik} : flow of material between facility i and facility k,

c_{jl} : cost per unit flow of material between location j and location l,

$$x_{ij} = \begin{cases} 1 \text{ if facility } i \text{ is at location } j, \\ 0 \text{ otherwise} \end{cases}$$

Then we can formulate the FLDP as a quadratic assignment problem:

$$\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ik}c_{jl}x_{ij}x_{kl} \tag{2.1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,2, \ldots, n,$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,2, \ldots, n, \tag{2.3}$$

$$x_{ij} \in \{0,1\}, \quad i, j = 1,2, \ldots, n. \tag{2.4}$$

The objective function (2.1) is quadratic in the variables x_{ij}'s. Constraints (2.2) ensure that each facility is given one location whilst constraints (2.3) ensure that each location is assigned a facility.

(ii) Quadratic Set Covering Problem

The general facility layout problem can also be modeled as a quadratic set covering problem (QSP) (Bazaraa (1975)). In this formulation, the total area occupied by all the facilities is divided into a number of blocks. Each facility is assigned to exactly one location and each block is occupied by at most one facility. By using the following additional notation:

q : number of blocks into which the total area occupied by all facilities is divided into,

$I(i)$: number of potential locations for facility i,

$J_i(j)$: set of block occupied by facility i if it is assigned to location j,

$d(j_i, l_k)$: distance between the centres of locations j and l if facility i is assigned to location j and facility k is assigned to location l,

$$p_{ijt} = \begin{cases} 1 \text{ if block } t \in J_i(j), \\ 0 \text{ otherwise,} \end{cases}$$

the quadratic set covering problem can be formulated as follows:

$$\text{Min} \sum_{i=1}^{n} \sum_{j=1}^{I(i)} a_{ij} x_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{I(i)} \sum_{k=1}^{n} \sum_{l=1}^{I(k)} f_{ik}\, d(j_i, l_k)\, x_{ij}\, x_{kl}, \tag{2.5}$$

subject to

$$\sum_{j=1}^{I(i)} x_{ij} = 1, \quad i = 1, 2, \ldots, n, \tag{2.6}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{I(i)} p_{ijt} x_{ij} \leq 1, \quad t = 1, 2, \ldots, q, \tag{2.7}$$

$$x_{ij} \in \{0,1\}, \quad i = 1, 2, \ldots, n, \tag{2.8}$$

$$j = 1, 2, \ldots, I(i).$$

In this formulation the objective function contains $x_{ij} \cdot x_{kl}$, which means that it can be classified as a quadratic function. Constraint (2.6) ensures that each facility is located in exactly one location, whilst constraint (2.7) ensures that each block is assigned by at most one facility.

(iii) Linear Integer Programming Problem

A suggestion to linearize the objective function of the model was given by Lawler (1963), who proposed the following formulation. Let

$$b_{ijkl}=\begin{cases} f_{ik}\,c_{jl}+a_{ij}, & \text{if } i=k \text{ and } j=l, \\ f_{ik}\,c_{jl}, & \text{if } i\neq k \text{ or } j\neq l, \end{cases} \tag{2.9}$$

Then the problem is

$$\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}\,x_{ij}\,x_{kl}, \tag{2.10}$$

subject to

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} x_{ij}x_{kl} = n^2, \tag{2.11}$$

$$x_{ij}+x_{kl}-2x_{ij}x_{kl} \geq 0, \quad i,j,k,l=1,2,\ldots,n, \tag{2.12}$$

$$x_{ij}x_{kl} \in \{0,1\}, \quad i,j,k,l=1,2,\ldots,n. \tag{2.13}$$

and the constraints (2.2)-(2.4).

Constraints (2.11) ensure that there are exactly n assignments in which plant i is located in location j, and n assignments in which plant k is located in location l. Note that a quadratic assignment problem with n^2 variables x_{ij} is linearized by defining n^4 variables y_{ijpq}, where

$$y_{ijkl} = x_{ij} \cdot x_{kl}.$$

This gives rise to the following formulation:

$$\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}\, y_{ijkl}\,, \tag{2.14}$$

subject to

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} y_{ijkl} = n^2, \tag{2.15}$$

$$x_{ij} + x_{kl} - 2y_{ijkl} \geq 0, \quad i,j,k,l = 1,2,\ldots,n, \tag{2.16}$$

$$y_{ijkl} \in \{0,1\}, \quad i,j,k,l = 1,2,\ldots,n. \tag{2.17}$$

and the constraints (2.2)-(2.4).

Constraints (2.15) ensure that there are exactly n assignments in which plant i is located in location j and n assignments in which plant k is located in location l. Constraints (2.16) ensure that the matrix is positive definite. These constraints represent a symmetric permutation of the rows and columns of one of the matrices.

(iv) Mixed Integer Linear Programming Problem

Kaufman and Broeckx (1978) formulated the facility layout design problem as a mixed integer linear programming problem by defining

$$w_{ij} = x_{ij}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}\, x_{kl}, \tag{2.18}$$

$$e_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}. \tag{2.19}$$

With this notation, we can write the objective function (2.10) as

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}\, x_{ij}\, x_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}\left(\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}\, x_{kl}\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \tag{2.20}$$

Hence, the problem can be restated as

$$\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \tag{2.21}$$

subject to

$$e_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}\,x_{kl} - w_{ij} \le e_{ij}, \tag{2.22}$$

$$w_{ij} \ge 0, \quad i,j = 1,2,\cdots,n, \tag{2.23}$$

and the constraints (2.2) - (2.4).

(v) Graph Theoretic Model

Using a graph theoretic approach, we can model the facility layout design as an edge-weighted graph in which the vertices represent the facilities and the edges represent the "adjacencies". The edge weight represents either the cost or the benefit of having two facilities adjacent. When the edge weight represents benefit, the problem is to find an arrangement which maximizes the total benefit. In this thesis, without loss of generality, we assume that our graph is complete (contains every edge). Thus our problem is feasible and the optimal solution will be a maximal planar graph, that is will consist of $3n$-6 edges.

We assume that the desirability of locating each pair of facilities adjacent to each other is known. Given a weighted graph G, the facility layout problem is to find a maximum weighted spanning subgraph G' of G that is planar. Mathematically, the problem is:

Maximize $\quad B(G) = \sum_{(i,j)\in E'} w_{ij}\,x_{ij}$

subject to

$x_{ij} = 0,1$, for all i,j,

and $G' = (V,E')$ is a planar graph,

where

$G=(V,E)$: is a weighted graph with V as a nonempty set of vertices (facilities), and E as a set of edges disjoint from V,

w_{ij} : the closeness rating (weight) indicating desirability of locating facility i adjacent to facility j,

$$x_{ij} = \begin{cases} 1 \text{ if facility } i \text{ is adjacent to facility } j, \\ 0 \text{ otherwise} \end{cases}$$

$$E' = \{(i,j) : x_{ij} = 1,\ (i,j) \in E\}.$$

The weights are presented in an adjacency matrix. Usually this matrix form is also called a **relationship chart** or a **relativity chart**. In an adjacency matrix form, the weight along the main diagonal is zero, as we assume there is no loop in the graph we construct. In our experiments, the entries in the adjacency matrix are edge weights, which are taken from generated random problems.

2.2 Exact Algorithms

In dealing with the facility layout problem some exact algorithms have been proposed (Gilmore (1962), Lawler (1963), Seppanen and Moore (1970), Foulds and Robinson (1976), Kaku and Thompson (1986), and Ketcham (1992)). Exact algorithms, which can give the best solution for a given problem, can be divided into two classes: **Branch and Bound** algorithms, and **Cutting Plane** algorithms.

2.2.1 Branch and Bound Algorithms

The Branch and Bound (B&B) method has been effectively used to solve a number of computationally difficult problems from the areas of combinatorial optimization. If the total number of feasible solutions for the problem being solved is small, we can investigate each feasible solution individually, and select the optimum by comparing them with each other. However, in most real life situations, such a method of total enumeration is impractical, as the number of solutions turns out to be very large.

B&B provides a method for searching an optimum solution by conducting careful enumeration, which hopefully is not total enumeration. The set of feasible solutions is partitioned into many simpler subsets. In each stage, a promising subset is selected and an effort is made to find the best solution for it. If the solution obtained is the best so far, then we update the bound. Otherwise we again partition this subset into two or more simpler subsets (called branching) and the same process is repeated until the best solution is obtained.

More precisely a general B&B algorithm can be stated as follows.

Step 1: Solve a relaxation (relax constraints or variable restrictions) of the original problem. This gives an upper bound. If the solution is integral, STOP.

Step 2: Create two new subproblems by branching on a fractional variable.

Step 3: A subproblem is no longer active when any of the following occurs:

1. The subproblem has been branched on;
2. All variables in the solution are integer;
3. The subproblem is infeasible;
4. The subproblem is fathomed (terminated) by a bounding argument.

Step 4: Choose an active subproblem and branch on a fractional variable. Repeat until there are no active subproblems.

The implementation of the branch and bound method is aided by the computation of a bound in each subset of the partition generated. These bounds are used in selecting promising subsets in the search for the optimum, and to discard some subsets that cannot possibly contain the optimum feasible solution. In minimization problems, lower bounds can be used to decide whether or not we do branching at a certain node in the search tree. If we have a lower bound for a subset that is greater than or equal to the current upper bound, there is no further branching from this node.

The success of the B&B method depends on the branching strategy, the search strategy, and the quality of the lower bound (in maximization) or upper bounds (in minimization) generated. Depth-first search strategy is often used as the search

strategy in the B&B method. In this search technique, a subset selected from the active list is explored until an improved feasible solution or a violation to the lower bound is found. Another subset is selected if the solution is infeasible. We then do branching from the parent node in the branch where exploration has not been done. Lower bounds can be obtained at each node by using a heuristic method. An alternate search in the B&B method is best-first, in which the subset with the lowest bound LB, is chosen to branch from.

B&B algorithms were first developed by Gilmore (1962) and Lawler (1963) for solving the QAP. These two algorithms assign a single facility to a single location, and compute all potential solutions and lower bounds. Different from Gilmore (1962) and Lawler (1963), Land (1963) and Gavett and Plyter (1966) proposed branch and bound algorithms by assigning pairs of facilities to pairs of locations. Pierce and Crowston (1971) proposed an algorithm which proceeds on the basis of stage by stage exclusion of pairs of assignments from a solution to the problems. Based on the concept of square matrix reduction, Burkard (1973) developed an exact algorithm for solving the QAP. In this algorithm, a matrix is transformed into another matrix so that the resulting matrix has only nonnegative elements and in each row and column there is at least one zero element. In the B&B algorithm developed by Bazaraa (1975), a partial layout at each stage is constructed and a bound for all possible completions of this existing partial layout is computed. The existing partial layout is then grown by assigning a new facility, provided that the selected assignment has a better benefit compared to the current bound. If the assignment results in a lower benefit, the forward search along this path is fathomed. If this is the case, a new different assignment has to be found. The process is carried out until the complete layout is obtained. Bazaraa and Elshafei (1979) proposed a branch and bound algorithm for the QAP, based upon the stage by stage assignment of single facilities to unoccupied locations.

Foulds and Robinson (1976) proposed a B&B algorithm for solving the facility layout design problem based on a graph theoretic approach. The algorithm is as follows. This technique starts with a list of non-increasing order of edge weights.

From the list, the $3n$-6 best-weighted edges are put in set A. The remaining edges (in order of weights) are put in set B. Then a tree, whose root represents a graph and whose nodes represent the combinations of choices of additional edges, is constructed. The edge with the highest weight in A is selected as the initial solution. To insert a new edge to the current subgraph, the possible high-weighted edge in set A is considered. If the addition of this edge does not violate planarity, insert this edge to the current partial subgraph. Otherwise select the highest-weighted edge from set B, and set the penalty to this selection. The penalty is the difference between the weight of the rejected edge from A and the weight of the selected edge from B. We branch from the node with least penalty, which has not finished its selection of $3n$-6 edges (after a planarity test is conducted). We can use the Planarity test proposed by Hopcroft and Tarjan (1974), or Foulds and Robinson (1976). Repeat until a maximal planar graph, constituted by $3n$-6 edges, is obtained. We continue until all nodes with a penalty less than that of the latest incumbent have been considered. The last incumbent is the optimal solution.

Burkard (1984) stated that the above exact algorithm has high memory requirements. Lavalle and Roucairol (1985) reported that parallel B&B algorithm requires high computational time for layout problems with 12 or more facilities. There are no computational results for the B&B method presented by Foulds and Robinson (1976). This algorithm, however, has been used by Eades *et al.* (1982) to analyze the performance of a heuristic, called the "Wheel Expansion" method (see Section 2.3.1). Problem instances which are solved optimally are of size 10.

2.2.2 Cutting Plane Algorithms

The cutting plane (CP) algorithm solves integer linear programs by modifying linear-programming solutions of relaxed problems until an integer solution is obtained. This algorithm refines a single linear program by adding new constraints successively so that the feasible region is reduced until an integer optimal solution is found.

The CP method starts by solving a continuous linear program relaxation. After a cutting plane (or more) is added to the linear program, this program is again solved in continuous variables by using the Simplex method. If the optimal solution to this new problem is integral, then we stop, as we have an optimal solution of the integer programming problem. Otherwise we again add a new cutting-plane (possibly more than one).

Suppose the original problem is P and the associated relaxed problem is P'. Let K be a set of valid inequalities for P. A lower bound for the original problem can be generated using the following procedure (Padberg and Rinaldi, 1991).

Cutting Plane Procedure:

Step 1. Set $L = \varnothing$.

Step 2. Solve P' with the additional constraints in L, and let x^* be the optimal solution.

Step 3. Find one or more inequalities in K that are violated by x^*.

Step 4. If none is found, stop. Otherwise add the violating inequalities to L and go to Step 2.

The above cutting plane procedure terminates when no further violation can be found or an optimal solution has been found. Suppose here we take a maximization problem. If an optimal solution is not obtained, then we subdivide the solution space by embedding the cutting plane procedure in a tree search technique. In this case, a check that the upper bound generated at Step 2 is higher than the best lower bound must be included. If the upper bound is at most equal to the best known lower bound, then the subproblem is fathomed. Step 4 is modified so that an integral solution, which satisfies all inequalities of K, becomes the best known solution and the lower bound is set accordingly. Note that if the above procedure is terminated when further violations can be found, then the objective function value is still a valid upper bound.

The success of a cutting plane method is heavily dependent on the efficiency of the cut generating technique and the quality of the cuts generated. Unfortunately, no systematic method is known to generate all (the linear equation or inequality) cuts, which define the feasible region containing the integer points.

Based on Bender's partitioning technique, Bazaraa and Sherali (1980) proposed a cutting plane algorithm. By using this technique, a mixed integer program is decomposed into its integer and continuous parts, each of which can be solved by an appropriate specialized algorithm in mathematical programming. The cutting plane algorithm is also used for solving QAPs (Burkard and Bonninger, 1983). There is no computational evidence available on the performance of cutting plane method available. Kusiak and Heragu (1987) indicated that the optimal cutting plane method has a high time and storage complexity. For example, the largest facility layout solved by using the cutting plane method is the layout problem with 8 facilities.

2.3 Heuristic Algorithms

We now focus our attention to heuristic procedures. The advantage of an exact algorithm is that it gives an optimal solution. Unfortunately, the computational complexity of the problem limits the size of the problem that can be solved. On the other hand, heuristic algorithms offer a solution, which is usually sub-optimal but computationally efficient. The performance of a heuristic algorithm can be evaluated by comparing their solutions with the optimal solution generated using an exact method or the bound obtained by taking the best $3n$-6 edges (n is the number of facilities). For the facility layout problem, the performance ratio is easily established as

$$\rho = \frac{B}{U} \times 100\ \%,$$

where B is the result obtained from the heuristic algorithm, and U is the upper bound computed from the best $3n$-6 edges.

To date, there are many heuristic procedures. These procedures are based on the methods of: graph theory; simulated annealing; expert systems; or tabu search. Since we will focus our attention on the graph theoretic based heuristics, we give more details of this approach and include only a brief discussion on the other approaches. We begin with the graph theoretic based heuristics.

2.3.1 Graph Theoretic Based Heuristics

A number of constructive heuristics that generate a maximal planar (weighted) subgraph have been proposed. Typically, these heuristics start with a K_3 or K_4 and build up the solution through vertex insertion, maintaining planarity at every stage. A major advantage of these methods is that there is no need to use a planarity test algorithm throughout the insertion process. Foulds *et al.* (1985) indicated that planarity tests, even though they can be easily implemented, do slow down algorithms in large applications.

In this section, we will focus our attention on a specific group of constructive heuristics, which do not require planarity testing. This comprises the main graph theoretic based heuristics developed to date: the Deltahedron Method (Foulds and Robinson, 1978), the Wheel Expansion Method (Eades *et al.*, 1982), the Green-Al Hakim Algorithm (Green and Al-Hakim, 1985), the Leung's Constructive Heuristic (Leung, 1992), TESSA (Boswell, 1992), and the Kim-Kim Algorithm (Kim and Kim, 1995).

To facilitate a comparative analysis of these heuristics, we give a brief description of them below. We begin with the Deltahedron Method.

(i) The Deltahedron Method (Foulds and Robinson, 1978)

The Deltahedron Method, which involves simple insertions, maintains a maximal planar graph at all times during the vertex insertions. Starting with an initial K_4, vertices are inserted one by one according to a benefit criteria. At each step, the

maximum benefit of inserting each unused vertex is calculated and the vertex yielding the highest benefit is selected and inserted into the current generated subgraph. In constructing an initial solution, a list of non-increasing order of edge weights is created. This is done by summing up the weight of edges incidents to each vertex. For each vertex i calculate

$$M(i)=\sum_{i\neq j} w(i,j).$$

The first four vertices are selected for an initial subgraph K_4. The next vertex in the list (according to the order) is then inserted into a face in the current partial subgraph for which the sum of the weights of the inserted edges is maximum. An example of 1-vertex (single) insertion operation is illustrated in Figure 2.1. In this operation, a vertex is inserted into a face, and is joined to all vertices of the face. This operation gives 3 new faces and removes 1 existing face. The process is continued until all vertices in the list are inserted into the existing graph. The resulting graph, which is maximal planar, is then the final solution. Using this construction, a solution can be generated in $O(n^2)$.

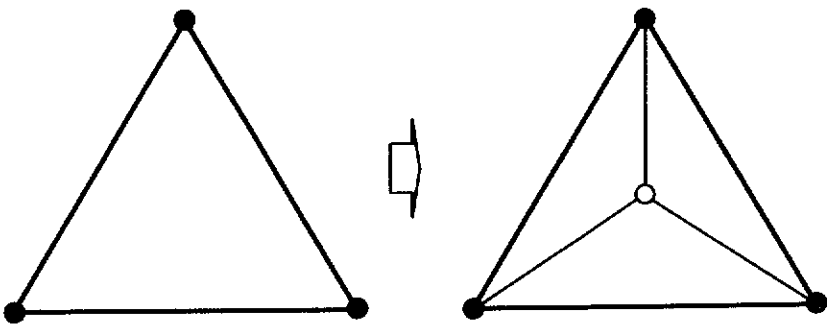


Figure 2.1. A single-vertex insertion into a face.

The above description is called **S-Construction**. Foulds and Robinson (1978) also describe another technique (called **R-construction**), which is more ambitious. The initial solution in this technique is obtained by generating all possible K_4's and

selecting the best. Once an initial solution is constructed, a vertex which yields the highest edge weight is inserted into an existing face in the current partial subgraph. This operation is carried out repeatedly until no further vertex insertion can be applied.

Foulds and Robinson (1978) tested the Deltahedron Method (S-Construction) on 6 small graphs ranging from 8 to 26 vertices. To illustrate the Deltahedron Method, consider the following example.

Example 2.1.

Consider the implementation of the Deltahedron Method (S-Construction) using the relativity chart in Foulds and Robinson (1978):

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
a	0	0	3	53	33	4	43	55	5	6
b	0	0	44	34	54	45	36	32	42	31
c	3	44	0	60	35	46	39	8	7	49
d	53	34	60	0	9	47	59	10	41	30
e	33	54	35	9	0	25	48	51	26	40
f	4	45	46	47	25	0	62	57	39	29
g	43	36	39	59	48	62	0	28	56	38
h	55	32	8	10	51	57	28	0	37	27
i	5	42	7	41	26	39	56	37	0	50
j	6	31	49	30	40	29	38	27	50	0

Table 2.1. A Relativity Chart of edge weights (size=10).

First we determine a K_4 by setting up a list of non-increasing order of weight. We compute the total weight of edges (i,j) incident to vertex i as follows.

M(a)	*M(b)*	*M(c)*	*M(d)*	*M(e)*	*M(f)*	*M(g)*	*M(h)*	*M(i)*	*M(j)*
202	318	291	343	321	354	409	305	303	300

Table 2.2. The sum of edge weights.

This table indicates that M can be arranged as

$$M(g) \geq M(f) \geq M(d) \geq M(e) \geq M(b) \geq M(h) \geq M(i) \geq M(j) \geq M(c) \geq M(a).$$

This means that the order of arrangement for insertion operation is

$$g, f, d, e, b, h, i, j, c, a.$$

The first 4 vertices (namely g, f, d, and e) are then taken as the initial solution, resulting in a tetrahedron with edges (d,e), (d,f), (d,g), (e,f), (e,g), and (f,g) and faces (d,e,f), (d,e,g), (d,f,g), and (e,f,g). Here (d,f,g) is an infinite (unbounded) face.

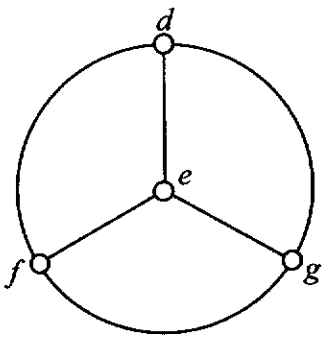


Figure 2.2. The initial solution (d,e,f,g).

Since b is the next vertex in the list, we consider vertex b to be inserted into one of the faces in the current partial subgraph. Below are the possible insertions and their corresponding weights:

Faces	(d,e,f)	(d,e,g)	(d,f,g)	(e,f,g)
Weight	133	124	115	135

Table 2.3. The insertion of vertex b and its possible additional weights.

Clearly the insertion of vertex b into the face (e,f,g) has the highest weight among the other possible faces. So we insert vertex b into this face, together with the corresponding edges (b,e), (b,f), and (b,g). This operation produces the following partial subgraph (partial solution).

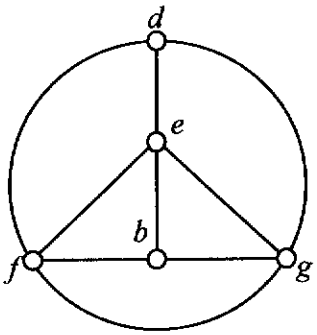


Figure 2.3. Inserting vertex b into the face (*e,f,g*).

The next vertex to be inserted is *h*. Again we compute all possible insertions and pick up an insertion which gives the heighest benefit. This computation is summarized in the following table.

Faces	(*b,e,f*)	(*b,e,g*)	(*b,f,g*)	(*d,e,f*)	(*d,e,g*)	(*d,f,g*)
Weight	140	111	117	118	89	95

Table 2.4. The insertion of vertex *h* and its possible additional weights.

Thus we select the face (*b,e,f*), since the insertion of vertex *h* into this face yields the highest weight 140. This gives us the partial subgraph (partial solution), illustrated in Figure 2.4.

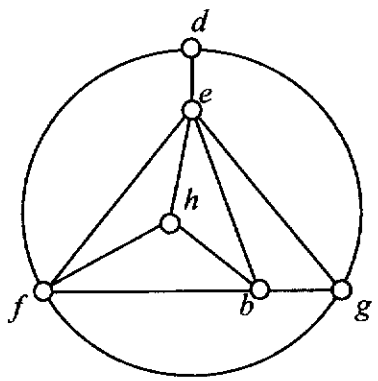


Figure 2.4. Inserting vertex *h* into the face (*b,e,f*).

We insert the remaining vertices, one by one, until the final solution consisting of 10 vertices is achieved. The following figure depicts the resulting final solution of weight 1055.

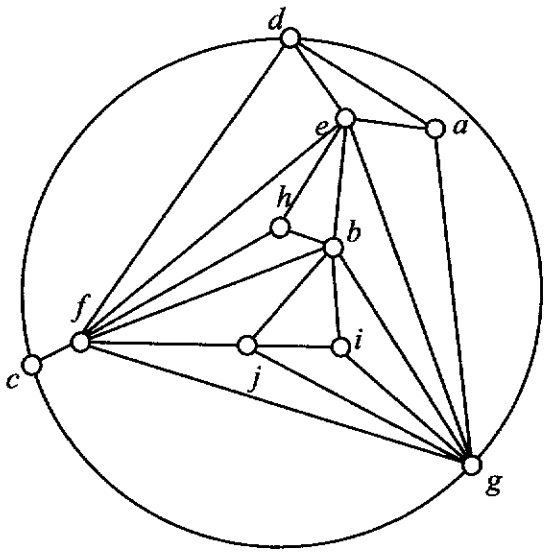


Figure 2.5. The final solution using S-Construction.

Now we consider the Deltahedron Method with R-Construction using the same set of edge weights. To construct the initial solution required in this method we generate all possible K_4's and select the best. From the set of data, we have a K_4 consisting of vertices *c*, *d*, *f*, and *g*, with the highest weight 313.

Next we consider 1-vertex insertion. If we compute all possible additional benefits produced by inserting a vertex into the existing faces, the highest additional benefit 145 is obtained if vertex *i* is inserted into the face (*d*,*f*,*g*). So we insert this vertex into the face (*d*,*f*,*g*) in the current partial subgraph. The next insertion operation is carried out in the similar fashion until all vertices are completely assigned. This gives the following final solution which has the weight 1063.

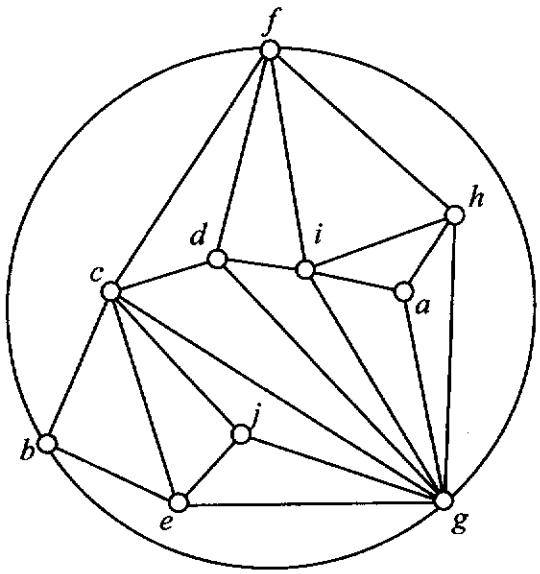


Figure 2.6. The final solution using R-Construction.

Foulds and Robinson (1978) also developed improvement techniques: edge interchange and vertex relocation. First we consider edge interchange operation, which they call Alpha operation (see Figure 2.7).

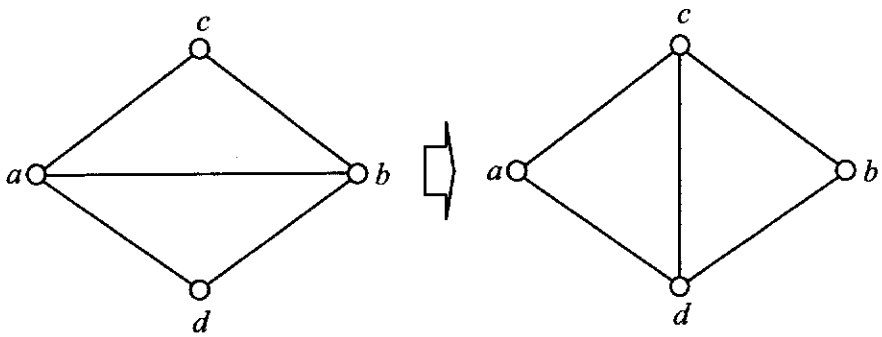


Figure 2.7. The process of Alpha Operation.

In this operation, an edge which is a common edge of a pair of faces is removed to create a quadrilateral and a new diagonal edge in this quadrilateral is then constructed. Suppose we have two faces (a,b,c) and (a,b,d) in which (a,b) as a common edge. If (c,d) is not already present, i.e. (a,b) is not braced, (a,b) can be removed and (c,d) is

then inserted to create a new pair of faces. This process will give a new construction of higher total weight if $w(a,b)$ is less than $w(c,d)$.

The complexity of this technique is $O(n^3)$ at each insertion step, although it can be reduced to $O(n^2)$ if we only check $(n\text{-}3)(n\text{-}4)/2$ prospective edges during the improvement procedures. Constructing the quadrilateral list, together with selecting prospective edges and selecting the best vertex relocation has a complexity of $O(n^2)$. The graph is maximal planar with n vertices, so there are exactly $3n$-6 quadrilaterals (each edge separates two triangular faces). This means at most $O(n)$ iterations are required and the total operation of improvements can be done in $O(n^3)$.

The Alpha operation can be applied to a maximal planar weighted graph. The operation may result in a graph of higher total weight, and thus improve the solution. However, this procedure has some limitations. It fails if the edge to be deleted is braced. The weight of the final solution given by the S-Construction (in Figure 2.5) can be increased by 8, if we replace edge (b,j) by edge (f,i). This is the only improvement we can do for the given solution. The other improvement cannot take place as either the edges are braced or the weight of the candidate edges are not better than the weight of the existing edges. The weight of the final solution produced by the R-Construction (in Figure 2.6) cannot be increased by using the Alpha operation, as there are no edges which can be interchanged. Based on computational results (using edge weights from random normally distributed with $\mu = 100$, $\sigma = 10$, and 20, and n=10, 15, 20, and 25, Al-Hakim (1991) claimed that the Alpha operation does not produce a superior solution in comparison with other improvement procedures.

Foulds *et al.* (1985) also proposed a vertex relocation technique, where a vertex of degree 3 is relocated to a new face in the existing graph. Let a be the vertex incident to edges (a,e), (a,f), and (a,g). Vertex a can be moved to a new face, face (x,y,z) say, so that edges (a,x), (a,y), and (a,z) are then constructed, while edges (a,e), (a,f) and (a,g) are deleted from the current solution. This operation can be carried out if $w(a,e) + w(a,f) + w(a,g) < w(a,x) + w(a,y) + w(a,z)$. There is no implementation of this technique reported.

(ii) Green-Al Hakim Algorithm (1985)

This algorithm, which is a slight variation of the Deltahedron Method, starts with a maximum weight K_3 (a triangle). This initial solution is obtained by selecting the highest weighted triangle from $n(n\text{-}1)(n\text{-}2)/6$ possible available triangles. Once the initial solution is found, then the 1-vertex insertion operation starts. The insertion technique is in the same fashion as that of the Deltahedron Method.

Example 2.2.

Suppose we implement the Green-Al Hakim algorithm to the edge weights in Example 2.1. Then we have (d,f,g) as the initial solution with the weight 168. Note that the second partial solution after the 1-vertex insertion is exactly the same as the initial solution in Example 2.1, i.e. (c,d,f,g) with the weight 313. Hence, the Green-Al Hakim algorithm produces the final solution for the given edge weights with the total weight 1063.

In this algorithm, a simple format is used to allow easy generation of the block layout plan corresponding to the solution. This step is taken by modifying the adjacency matrix of relativity chart to represent a graph. Its interpretation is as follows:

$$a_{ij} = \begin{cases} 1 \text{ for } (i,j) \text{ in } E(G) \\ 0 \text{ otherwise.} \end{cases}$$

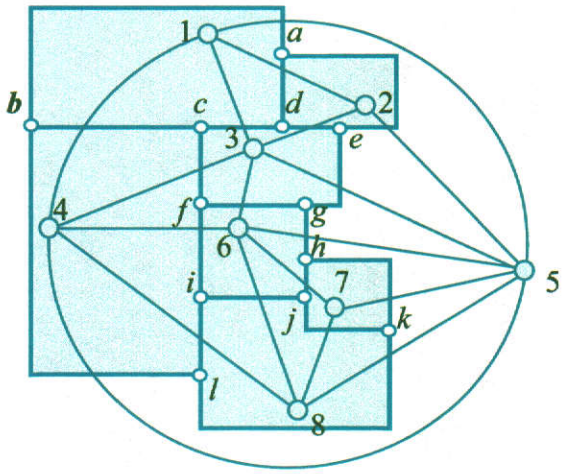
This formulation is used to create M-Matrix: $M = \{m_{ij}\}$, where

$$m_{ij} = \begin{cases} (u,w) \text{ for } a_{ij} = 1, (u,w) \text{ in } E^*, \ (i,j) \text{ intersects } (u,w) \\ 0 \quad \text{otherwise} \end{cases}$$

and E^* is the set of edges in G^*, where G^* is a dual graph of G.

For each non-zero entry a_{ij}, there is a corresponding edge (u,w) of the layout graph in the block plan. Here m_{ij} replaces a_{ij} in matrix A. The resulting matrix containing m_{ij} is called "M-Matrix". Since this matrix is symmetric, we simply deal

with the upper part triangle or the lower part triangle of the matrix (excluding the main diagonal). Using the *M*-Matrix a layout graph can be easily constructed. An example of an *M*-Matrix together with the corresponding graph and its layout is illustrated in Figure 2.8.



	1	2	3	4	5	6	7	8
1	---	(*a*,*d*)	(*c*,*d*)	(*b*,*c*)	(*a*,*b*)	---	---	---
2		---	(*d*,*e*)	---	(*a*,*e*)	---	---	---
3			---	(*c*,*f*)	(*e*,*g*)	(*f*,*g*)	---	---
4				---	(*b*,*l*)	(*f*,*i*)	---	(*i*,*l*)
5					---	(*g*,*h*)	(*h*,*k*)	(*k*,*l*)
6						---	(*h*,*j*)	(*i*,*j*)
7							---	(*j*,*k*)
8								---

Figure 2.8. A graph and its corresponding *M*-Matrix.

There is no information available on the relative performance of the algorithm, so the quality of solution is not known. We investigate its performance along with other heuristics in Chapter 3. The insertion technique implemented in this algorithm is the same as that of the Deltahedron Method, and so the complexity of this algorithm is $O(n^3)$.

Al-Hakim (1991) proposed an improvement technique called the **Gamma operation** (Γ-operation), which can be used to replace all edges of maximal planar graphs. In this operation, any edge, whether braced or unbraced, may be replaced by another edge. The improvement procedures are carried out by interchanging non zero and zero entries of the M-matrix. Suppose H is the highest weight associated with the zero entries in the M-matrix, and N is the set of non-zero entries in the M-matrix that have weights less than H. To improve the solution, we interchange any element of N with a higher weight zero entries. Both improvement procedures have complexity $O(n^3)$. This improvement can be conducted through the following procedures:

Procedure 1:

Check the entries of N for possible replacement with a zero entry in M. Carry out the Gamma operation which results in the highest additional weight until no further improvement can be made.

Procedure 2:

Construct a set of non-zero entries in N that can replace the prospective zero entry. Select the non-zero entry with the least weight for replacement, and continue until no further reduction in H can be made.

The Gamma operation, which takes place in the above procedures, has 3 possible cases:

Case 1:

This case is the same as Alpha operation of Foulds and Robinson (1978).

Case 2:

This operation is implemented when (a,b) is braced by (c,d), and the edge bracing (c,d) contains a vertex which is also an end vertex of (a,b). The subgraph containing

this operation is a subgraph of 5 vertices. There are two possible subgraphs having this feature. Figure 2.9 displays these two graphs and their corresponding graphs after the operation is applied. In this operation, (a,b) is replaced by (b,e). Note that the short lines incident to a vertex in these graphs indicate that there maybe other edges emanate from the vertex.

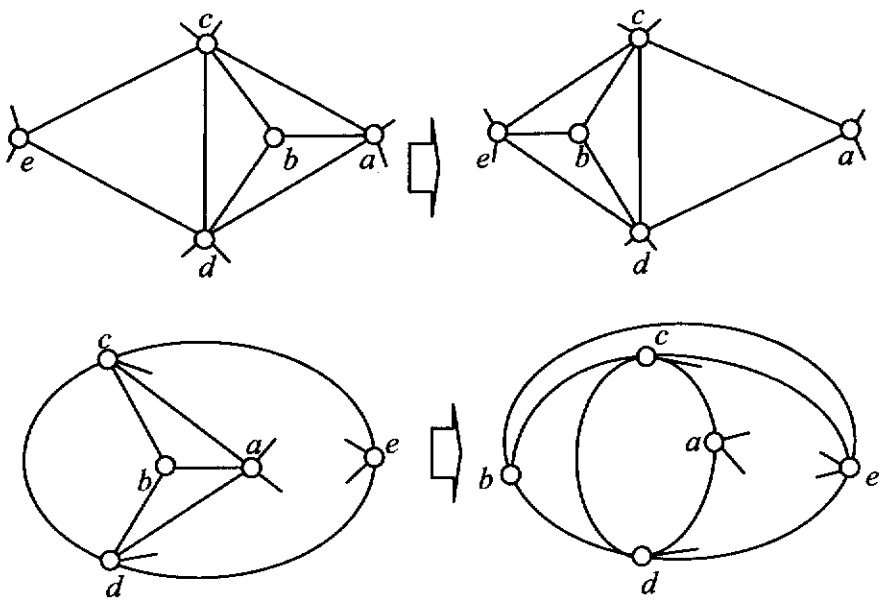


Figure 2.9. Diagonal operations on braced graphs (Case 2).

Case 3:

Case 3 is applied when a braced edge already exists in the graph so that the implementation of the procedures in cases 1 and 2 are redundant. Figure 2.10 depicts a subgraph which contains (a,b), the edge that will be removed. The only possible candidates to be inserted for replacement of (a,b) are (a,f), (b,e), and (e,f). In the same figure, 3 resulting subgraphs containing these three new edges, are also displayed. Note, that the edge (a,b) is no longer available in these subgraphs.

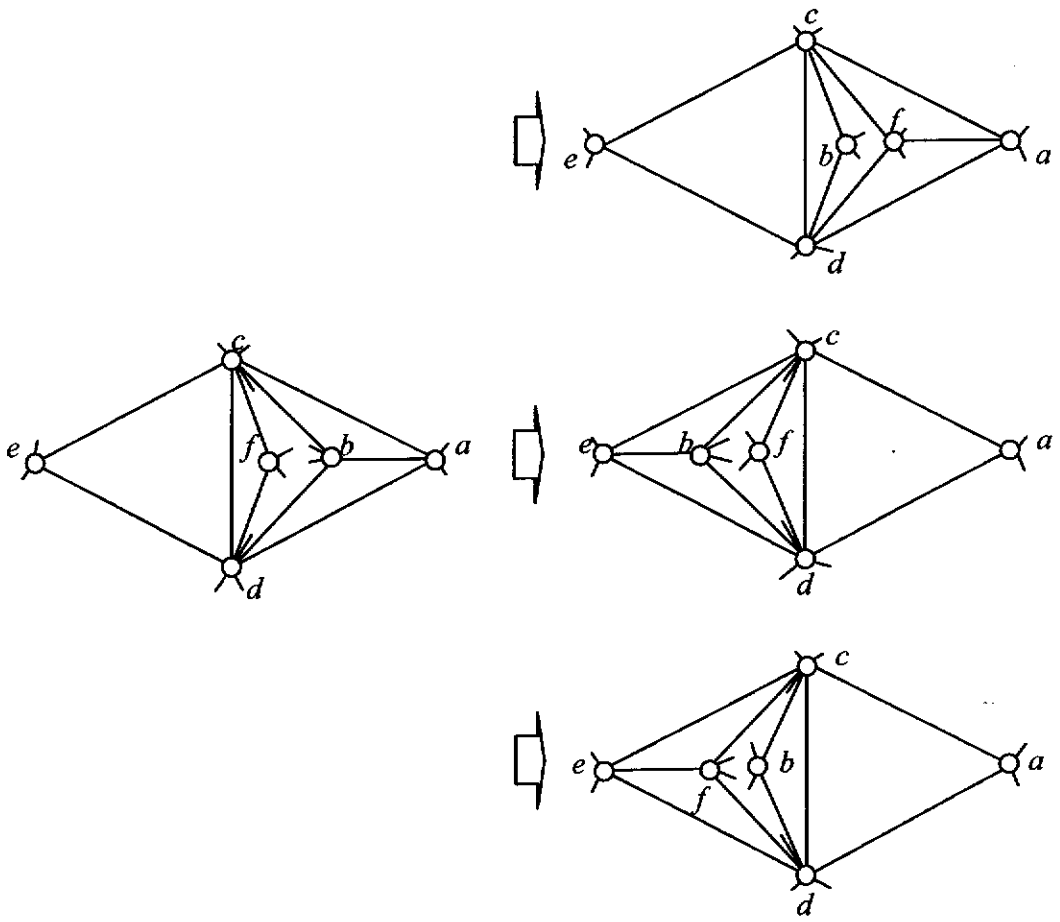


Figure 2.10. Diagonal operations on braced graphs (Case 3).

Computational results are based on problems of size n = 10, 15, 20, and 25. The weights for the REL Chart chosen randomly from normal distribution with mean μ= 50 and σ=10 and 20. The solution quality S is formulated as S =100(improved solution)/initial solution. From the computational results, Procedure 1 and Procedure 2 with Alpha operation and Gamma operation produce improvements as displayed in Table 2.5. The third column depicts the lowest and the highest solution quality for each procedure when only Alpha operation takes place, whilst the fourth column represents the solution quality for each procedure when Gamma operation (including Alpha operation) is implemented.

Procedure 1:

n	σ	Alpha Operation (%)	Gamma Operation (%)
10	10	0.00 – 4.12	5.26 – 7.04
	20	1.68 – 4.34	5.71 – 7.92
15	10	2.01 – 4.37	4.76 – 7.93
	20	3.01 – 3.64	6.17 – 8.62
20	10	4.01 – 5.99	6.01 – 8.64
	20	4.15 – 6.01	5.80 – 8.13
25	10	3.25 – 7.05	6.12 – 8.60
	20	4.10 – 6.75	7.01 – 9.10

Procedure 2:

n	σ	Alpha Operation (%)	Gamma Operation (%)
10	10	0.00 – 3.92	5.37– 7.04
	20	2.01 – 2.75	5.98 – 8.14
15	10	2.11 – 4.72	4.57 – 8.15
	20	1.92 – 3.47	6.66 – 8.62
20	10	3.02 – 5.70	6.12 – 6.25
	20	3.16 – 5.01	5.61 – 9.10
25	10	4.01 – 7.25	5.99 – 9.03
	20	3.80 – 6.98	7.62 – 9.02

Table 2.5. The performance comparison of the two procedures (Al Hakim, 1991).

From these tables we can see that the improvement can be higher if the Gamma operation (which also includes the Alpha operation) is used in both procedures. In his computational results Al Hakim (1991) also presented computational time, which indicates that Procedure 2 requires less time than Procedure 1. Using Gamma operation, for n=10, 15, 20, and 25, Procedure 1 requires average time 5.98, 9.25, 17.34 and 27.13 seconds, respectively. Similarly, Procedure 2 (for the same n) requires average time 4.29, 6.73, 13.24, 20.58 seconds, respectively.

(iii) The Leung's Constructive Heuristic (Leung, 1992b)

This heuristic can also be viewed as a generalization of (i) in that at each step either one or three vertices are inserted. Starting with K_4 as its initial solution, the solution is grown by inserting 3 vertices or 1 vertex into a face in the current partial graph that can maximize the additional edge-weight per vertex added.

In each insertion process the resulting graph is always planar, since a triangulation operation is always carried out. The single insertion option is evaluated as in (i). The triple vertex insertion involves 9 new edges, and the benefit of the insertion is evaluated by dividing the sum of the weights of these 9 edges by 3. The insertion of triple vertices is depicted in Figure 2.11.

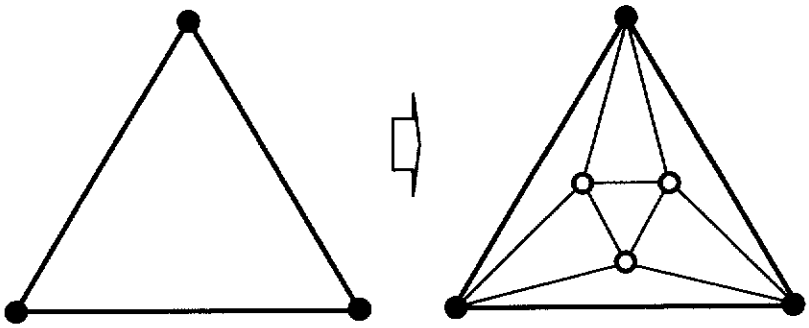


Figure 2.11. A triple-vertex insertion into a face.

There are 6 different possible subgraphs produced if 3 vertices are inserted into a face. Suppose the existing face has vertices a, b, and c, whilst the new inserted vertices are x, y, and z. Figure 2.12 displays all the possible subgraphs produced by 3-vertex insertion operation.

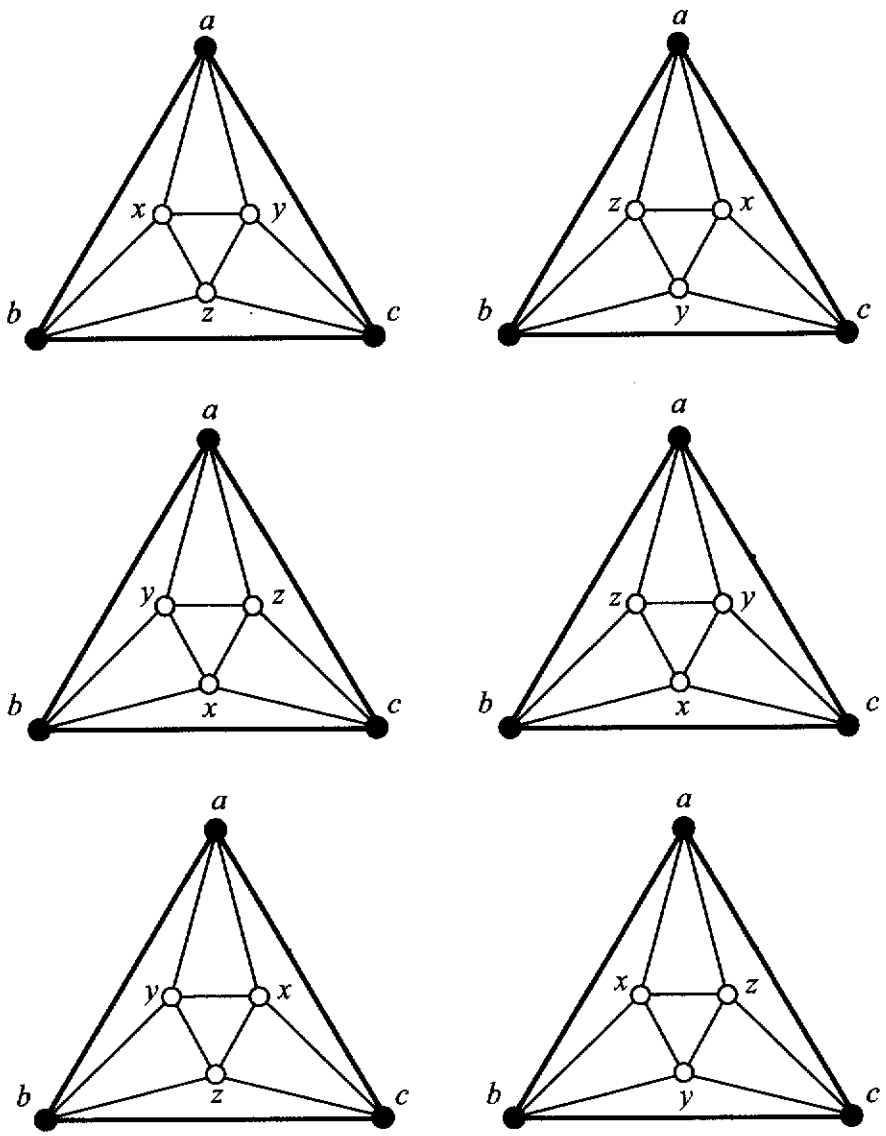


Figure 2.12. Six possible subgraphs in triple vertex insertion.

Example 2.3.

We illustrate this algorithm by applying it to the edge weights presented in Example 2.1. Below are the partial solutions produced successively using the insertion operations in this method.

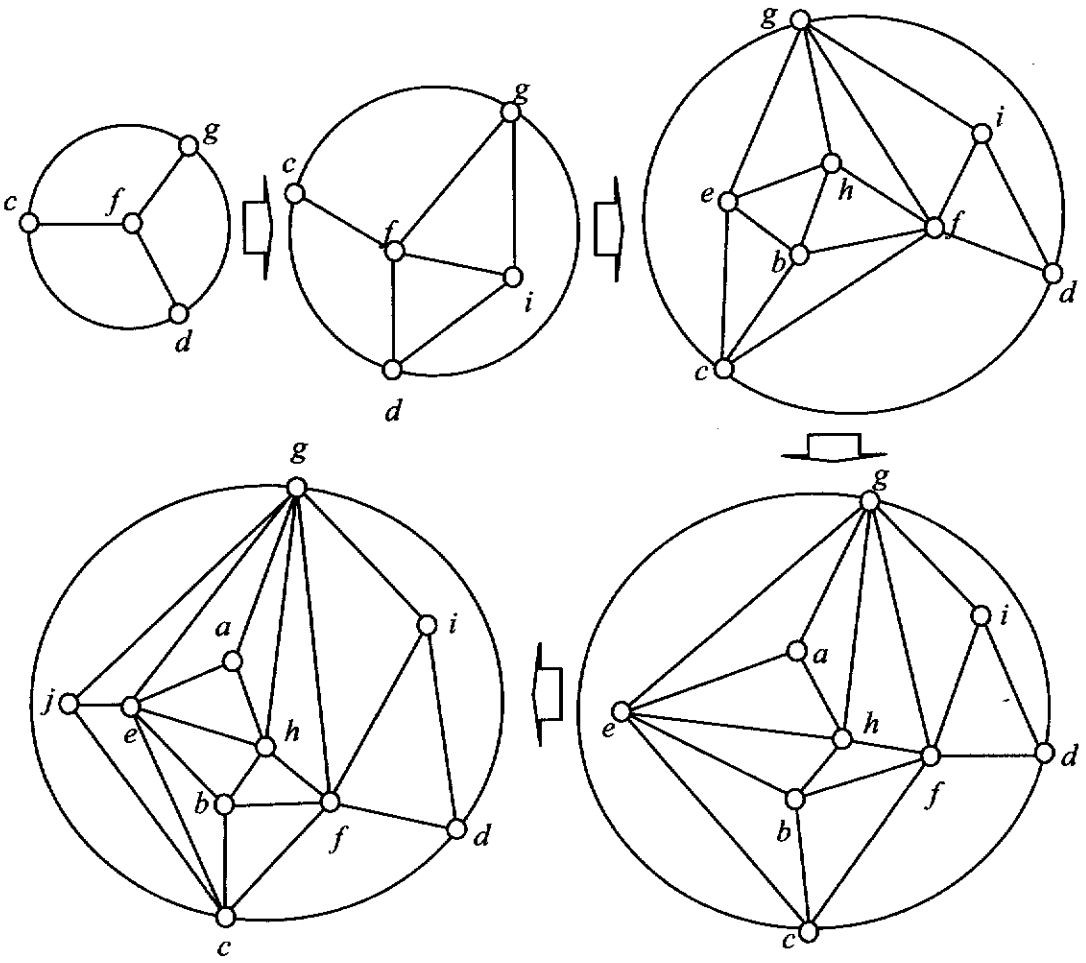


Figure 2.13. Insertion operations in Leung's Constructive Heuristic.

Computing all the possible K_4's, we have (c,d,f,g) as the highest-weighted initial solution, which has weight 313. The weight computation gives the highest additional benefit of 145 if vertex i is inserted into the face (d,g,f). Next, the additional weight 304 is obtained when vertices b, e, and h are inserted into the face (c,f,g). Vertex a is the next vertex to be located in the face (e,g,h) which contributes the benefit of 133. The last vertex is j, which is inserted into the face (c,e,g). This yields additional benefit of 127; and hence the final solution has weight 1101.

Although the Leung's Constructive Heuristic usually generates a better solution than (i), the processing time is considerably longer. The complexity of the

algorithm is $O(n^5)$. This algorithm was tested using a set of 90 problems; values of n used were 20, 30, and 40. The edge weights were taken from a normal distribution with mean 100 and standard deviation 5, 10, 15, 20, 25, and 30.

(iv) Wheel Expansion Algorithm (Eades *et al.*, 1982)

Here the initial K_4 is obtained by selecting an edge, having the highest weight, and then applying two successive vertex insertions according to the benefit criteria. The algorithm then proceeds with an insertion process, called the Wheel Expansion Procedure.

A **wheel** on n vertices is defined as a cycle on $(n\text{-}1)$ vertices (termed the **rim**), such that each vertex is adjacent to one additional vertex (termed the **hub**). See Figure 2.14.

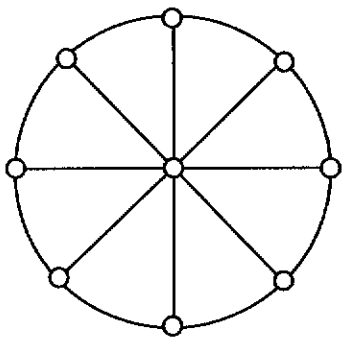


Figure 2.14. A wheel on 8 rims.

Let W be a wheel having the hub x. Select two vertices k and m, which are on the rims of this cycle. A vertex y from the set of unused vertices is then inserted into the wheel in current partial subgraph, such that y becomes a hub of the new wheel W' containing k, m, and x as its rims, and all rims in W are now adjacent to vertex x or vertex y. By inserting each unused vertex successively using the above fashion, the final maximal planar subgraph is obtained. See Figure 2.15.

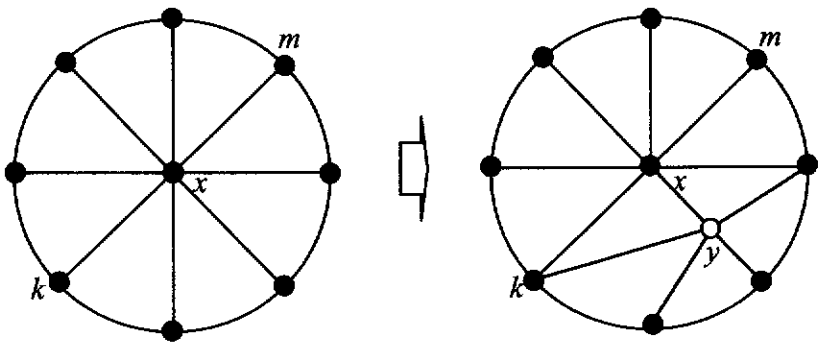


Figure 2.15. A vertex insertion in the Wheel Expansion Method.

Example 2.4.

Applying the Wheel Expansion Method to the edge weights in Example 2.1 gives us an initial solution (*c*,*d*,*f*,*g*) with the weight 313. The following figure displays the initial solution and the second partial subgraph, followed by the final solution which has the total weight 1101.

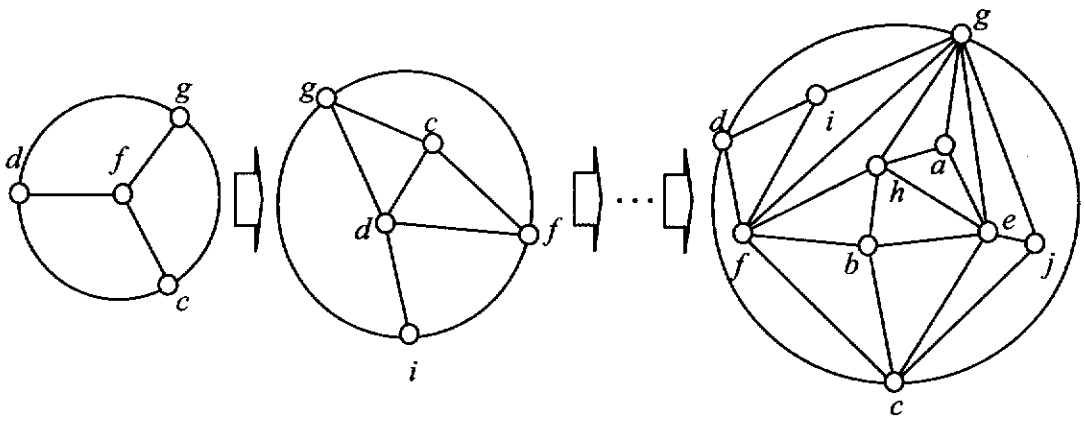


Figure 2.16. Insertion operations in the Wheel Expansion Method.

Eades *et al.* (1982) tested their algorithm on graphs with n=10, 20, and 30. The edge weights were chosen randomly from a normal distribution with mean μ = 100 and standard deviation σ = 5, 10, 15, and 20. The performance of the algorithm for n=10 is compared to that of the optimal solution (obtained by an exact method). The

performance of the algorithm is also compared to the best value of $3n$-6 edges (as the upper bound), for n=10, 20, and 30. The following tables display the summary of the results. Note that H represents the total weight found by the Wheel Expansion Method, W^* represents the optimal total weight found by the exact method, and B represents the best value of $3n$-6 edges.

N	σ	$100H/W^*$ (%)
10	5	99.2 - 99.8
	10	98.8 – 99.4
	15	96.6 – 98.6
	20	95.3 – 99.7
	25	95.1 – 98.2

Table 2.6. The Wheel Expansion Solution against the Optimal Solution.

N	σ	$100H/B$ (%)
10	5	97.9 – 98.4
	10	95.9 – 98.0
	15	94.8 – 95.9
	20	92.8 – 94.9
	25	91.4 – 93.5
20	5	97.5 – 98.0
	10	95.0 – 95.8
	15	92.2 – 94.8
	20	91.7 – 93.2
	25	89.8 – 91.6
30	5	97.3 – 97.5
	10	95.2 – 96.0
	15	92.8 – 94.4
	20	90.7 – 92.0
	25	89.6 – 90.2

Table 2.7. The Wheel Expansion Solution against the Upper Bound.

These tables indicate that the algorithm shows good solution for low variance. Compared to the optimal solution, for n=10, the solution quality reaches 99.8% (Table 2.6). Compared to the best $3n$-6 edges, for n=10 and low variance, the solution quality

is in the range 97.9 – 98.4%, and at the highest variance it reaches 91.4%. For n=30, with low variance, the solution quality reaches 97.5% and for the high variance it reaches 90.2% (Table 2.7).

The complexity of this algorithm is $O(n^4)$ in the worst case. However, in randomly generated graphs whose vertex degrees are bounded, the computational complexity is $O(n^3)$.

(v) Kim-Kim Algorithm (Kim and Kim, 1995)

This algorithm is only a slight variation of (iii). It starts with the highest-weighted K_4. Here, instead of considering 1 vertex or 3 vertices in each iteration to obtain the highest weight insertion, it always considers 3-vertex insertion until there are two vertices or one vertex left in the set of unused vertices. The complexity of this algorithm is $O(n^4)$.

Example 2.5.

Now we again use the edge weights in Example 2.1. The initial solution for this problem is the same as that of the Leung's Constructive Heuristic. In Figure 2.17 the whole process of insertion operations are displayed. This gives rise to the final solution which has the total weight 1082.

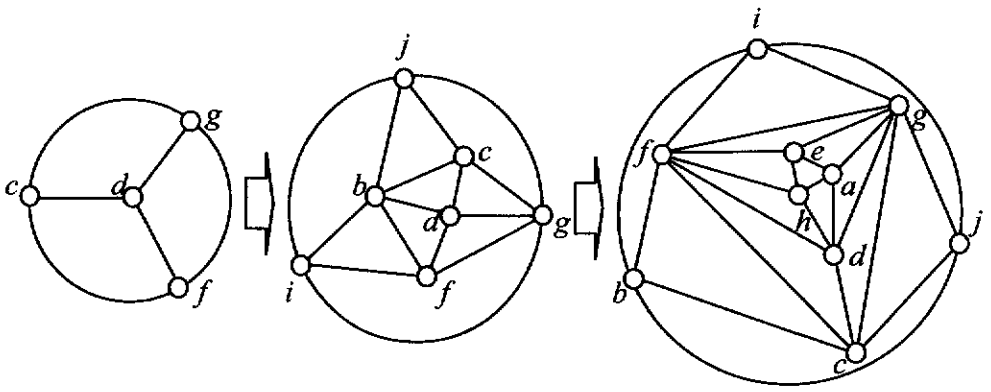


Figure 2.17. Insertion operations in Kim-Kim Algorithm.

The Kim-Kim algorithm also incorporates improvement procedures comprising the vertex interchange and the vertex movement. Note that the vertex movement here is the same as that of Foulds and Robinson (1978). In the vertex interchange procedure two vertices are interchanged together with their corresponding edges. In vertex movement, a vertex with degree 3 is deleted from the current subgraph and relocated in a new face of the graph. Since this movement is only suitable for a vertex of degree 3, other vertices with degree more than 3 cannot be moved by using this technique.

The vertex movement and the vertex interchange can be implemented simultaneously or implemented in order. So, there are 4 possible options:

a. Vertex interchange (called INHE).
b. Vertex movement and vertex interchange simultaneously (called SIM).
c. Vertex interchange followed by vertex movement (called MOIN).
d. Vertex movement followed by vertex interchange (called INMO).

The above procedures were tested on 100 problem instances that are generated randomly with the size n=8, 12, 15, 20, and 30. Kim and Kim compared the above procedures with CRAFT (Buffa *et al.*, 1964), an algorithm for solving FLDP. In CRAFT, the overall material flow data costs for an initial block layout is calculated. The algorithm scans all possible potential savings in the objective function made possible by exchanging the location of the two departments and selecting that exchange which results in the largest saving. After this particular exchange is effected, the new block layout is printed, its overall handling cost computed and the same steps are repeated. The algorithm is completed when successive steps fail to reduce the value of the objective function. From the computational results, the performance of the above procedures, except for INHE, outperform CRAFT.

(vi) TESSA (Boswell, 1992)

TESSA starts with an initial K_3, chosen according to weight. It considers only triangular faces and at each step a list of available triangles is maintained. The selected face is chosen according to weight and must have at least one edge in common with the boundary of the current partial subgraph. Note that at each step either a single edge or a vertex and two edges are added (see Figure 2.18). If the selected face contains an edge or a vertex, which already exists in the interior, this face is rejected. Faces which have no common edges with the boundary are rejected from consideration, but they might be reconsidered in the future insertion. In each iteration, except in the final stage, the resulting graph may not be maximal planar.

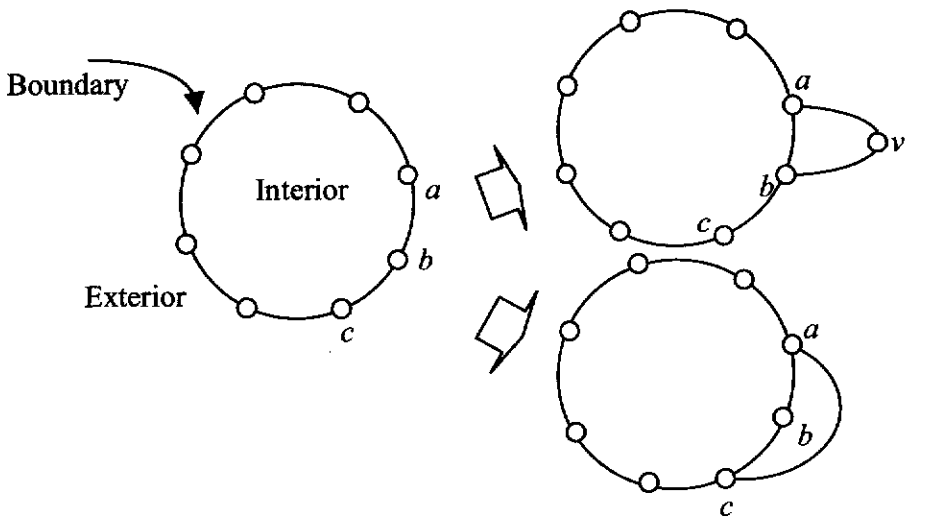


Figure 2.18. Face additions in TESSA.

Example 2.6.

Consider the edge weights in Example 2.1. Applying TESSA to this set of data gives the initial solution (d,f,g) of weight 168. The final solution, which has weight 1059, is illustrated in Figure 2.19.

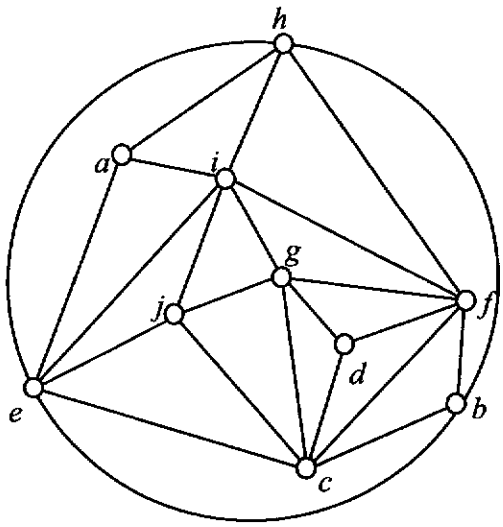


Figure 2.19. The final solution by using TESSA.

TESSA was implemented and tested using randomly generated data for each of n=10, 20, 30, and 40. The edge weights are normally distributed with a mean μ = 100 and standard deviations σ = 5, 10, 15, 20, 25, and 30. The results are presented in Table 2.8. In this experiment, compared to the upper bound (the best $3n$-6 possible edges), the best performance of the algorithm is 99.2% (n=10, σ=5), and for the higher variance it just reaches 88.6%.

n	Standard deviation (σ)					
	5	10	15	20	25	30
10	98.2-99.2	97.9-99.5	93.7-97.6	94.5-97.6	93.5-96.6	93.9-97.1
20	98.2-98.7	96.3-97.3	93.5-96.6	92.2-95.2	92.2-94.6	90.0-95.7
30	97.5-98.2	96.0-96.2	92.7-94.6	92.1-94.1	90.9-93.0	90.1-94.1
40	97.6-98.1	95.0-95.6	93.1-94.7	89.8-93.2	91.0-93.2	88.6-90.8

Table 2.8. The performance of TESSA (% of upper bound).

Boswell (1992) stated that this algorithm, which has the complexity $O(n^5)$, requires significantly more computational time than the others. Wascher and Merker (1997) also mentioned, that TESSA is the slowest method of all; therefore it may not be applicable for large problems. Watson and Giffin (1996), using the worst case analysis of heuristics which is due to Dyer *et al.* (1985), have shown that TESSA will perform arbitrarily badly in the worst case, although test results on average problems are promising.

Although Boswell showed that the heuristic performance is very good on the test problems, Al-Hakim (1994) argued by stating that TESSA would generate an infeasible solution. Watson and Giffin (1996) states that Al-Hakim is not entirely correct, and the solution is feasible although it has a very poor total weight.

(vii) The String Processing Algorithm

Seppanen and Moore (1975) and Moore (1976) introduced an algorithm for solving the facility layout problem, where string symbols corresponding to a graph are manipulated to obtain the solution. So, whilst this algorithm is based on graph theory, it uses string language. In the procedure, planarity is always maintained throughout the insertion process, and so there is no planarity testing required.

To apply the algorithm, first we construct a **maximal spanning tree** (a subgraph of a tree) using Kruskal's algorithm. It is then followed by generating symbol string languages to represent a class of planar graphs and trees. Let $G=(V, E)$ be a finite graph of n vertices. A **rewriting rule** in string language is a relation of the form $\phi \rightarrow \varphi$, where ϕ and φ are strings of symbols from V. This means that if a string ϕ is generated, then the string φ can also be generated.

Tree Boundary

Let $T=(V,E^*)$ be a tree of n vertices. The **boundary** of T is defined as a string of symbols obtained as follows. We take any vertex in T as a starting point. The

symbol of the vertex is taken as the first symbol of the boundary string. To obtain the rest of the string, the tree is traversed in a selected direction (clockwise or anti clockwise). When a vertex is traversed, its symbol is inserted into the string. The traversal is continued until it reaches the starting point.

The Grammar of Trees and Planar Graphs

To grow the tree and to obtain a planar graph by implementing a string manipulation, we use several rules. For initiating the tree we start with a single edge using the following rule.

Rule 1: $T \rightarrow ab$.

Once a tree is planted, then for any existing symbol c in the tree and any new symbol x, the tree can be grown by using Rule 2:

Rule 2: $c \rightarrow cxc$.

To construct a planar graph, a number of edges are added to the existing spanning tree, by embedding the edge to the planar representation of the graph. Suppose T is a tree and $S = aAbB$ is a string generated by G_T which represents the boundary of T with two "distinguished" entries a and b. The symbols a and b are said to "delimit" the substrings A and B in the circular boundary string S. We can convert the string into two different strings, using a rewriting rule as follows.

Rule 3: $\{aAbB \rightarrow aAb\ bBa\}$

This rule partitions the string S into 2 different strings, each of them contains a substring which is still bounded by the "delimiting" symbols.

Example 2.7.

Consider the tree in Figure 2.20.

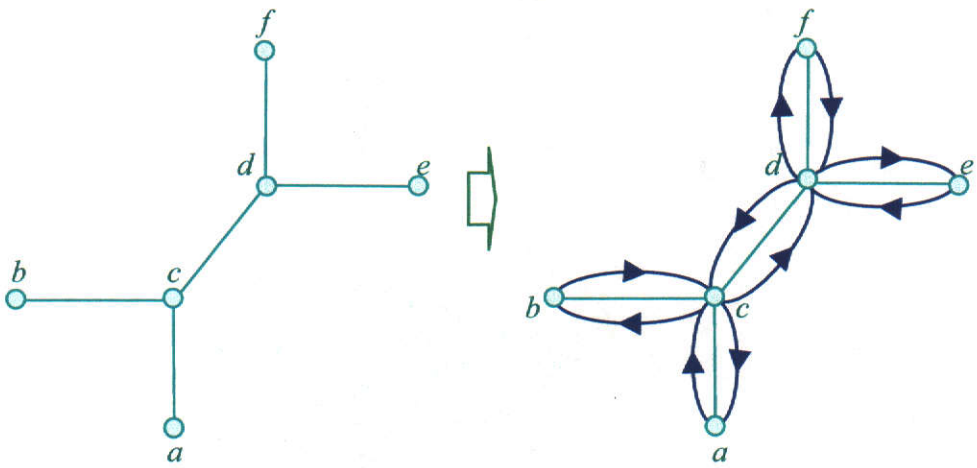


Figure 2.20. A traversed tree and its boundary.

By defining vertex a as an initial point, we can express the boundary of the tree as a string $S = acbcdfdedc$.

Example 2.8.

Suppose we have a spanning tree T illustrated in Figure 2.21.

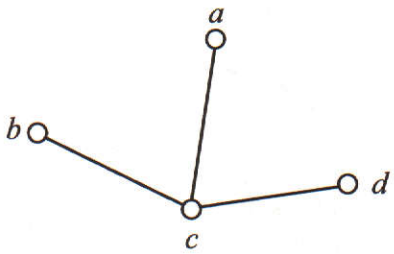


Figure 2.21.

Suppose we take vertex c as a starting point and the edge (c,b) as the initial tree. To obtain a tree boundary from this tree we generate a string as follows (the bold font represents the vertex as a centre point).

$$T \rightarrow \boldsymbol{c}b$$
$$\rightarrow c\boldsymbol{a}cb$$
$$\rightarrow cd\boldsymbol{c}acb$$

To generate a maximal planar graph by using a string representation we partition this string into several separate different strings as follows (delimiting symbol strings, which represent the vertex and join two vertices, are in bold font):

cdcacb → *cdcab acb* (Joining vertex *a* to vertex *b* gives the face (*a*,*c*,*b*))

→ *cdb dcab acb* (Joining vertex *d* to vertex *b* gives the face (*c*,*d*,*b*))

→ *cdb dab dca acb* (Joining vertex *d* to vertex *a* gives the faces (*d*,*a*,*b*), and (*d*,*c*,*a*)).

The final row represents a maximal planar graph consisting of faces (*c*,*d*,*b*), (*d*,*a*,*b*), (*d*,*c*,*a*), and (*a*,*c*,*b*). Here we assume that the order of edge selection is based on edge weights. In this example the order is (*a*,*b*), (*d*,*b*), (*d*,*a*).

Example 2.9.

To understand how the String Processing algorithm works, we take an example of 6 vertices using edge weights from Boswell (1992).

	a	***b***	***c***	***d***	***e***	***f***
a	0	55	45	45	83	91
b	55	0	78	38	72	16
c	45	78	0	52	43	69
d	45	38	52	0	85	41
e	83	72	43	85	0	50
f	91	16	69	41	50	0

Table 2.9. Edge weights in a Relativity Chart (Boswell, 1992).

First we construct a maximal spanning tree using Kruskal's algorithm. We put the edge weights in a list of non-increasing order (see Table 2.10).

	Edges	Weight
1	(a,f)	91
2	(d,e)	85
3	(a,e)	83
4	(b,c)	78
5	(b,e)	72
6	(c,f)	69
7	(a,b)	55
8	(c,d)	52

	Edges	Weight
9	(e,f)	50
10	(a,c)	45
11	(a,d)	45
12	(c,e)	43
13	(d,f)	41
14	(b,d)	38
15	(b,f)	16

Table 2.10. The list of non-increasing order of edge weights.

Based on this order and the weight of each edge, we have the following maximal spanning tree: $(a,f) \rightarrow (a,e) \rightarrow (d,e) \rightarrow (b,e) \rightarrow (b,c)$.

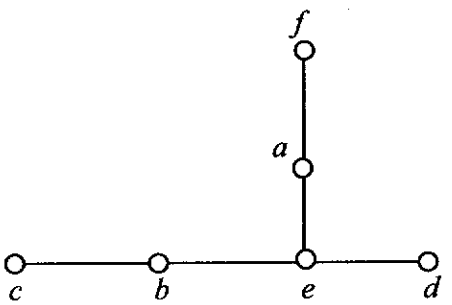


Figure 2.22. A maximal spanning tree.

To construct a string from the above maximal spanning tree, we need to select a vertex, and take a string representing this vertex as a starting point. Without loss of generality, we take a and start from the string *af* (which represents the edge (a,f)).

$af \rightarrow aeaf$	(obtained by adding *ae*)
$\rightarrow aedeaf$	(obtained by adding *de*)
$\rightarrow aedebeaf$	(obtained by adding *be*)
$\rightarrow aedebcbeaf$	(obtained by adding *bc*).

Now we construct a maximal planar graph by triangulating this tree. We start by inserting the highest-weighted edge (c,f) into the existing maximal spanning tree. To do this, we break the string $S = aedebcbeaf$ into $S_1 = cbeaf$ and $S_2 = aedebcf$. Based on its weight, (a,b) is the next edge to be inserted. From S_1 we get a new string $S_3 = bea$ and $S_4 = cbaf$. The next selection is (c,d). So we embed cd on the string S_2 which results in $S_5 = aedcf$ and $S_6 = debc$. Embedding ef on S_5 gives $S_7 = edcf$ and $S_8 = aef$, and embedding ca on S_4 produces $S_9 = cba$ and $S_{10} = caf$. The next selection is df, so we embedd it on S_7, resulting in $S_{11} = dcf$, and $S_{12} = edf$. The last embedding is bd on S_6, which produces $S_{13} = deb$ and $S_{14} = dbc$.

From the above process we obtain the following faces which constitute a maximal planar graph as the solution:

(b,e,a), (c,b,a), (c,a,f), (a,e,f), (d,c,f), (e,d,f), (d,e,b), and (d,b,c).

The picture of this maximal planar graph, with the total weight 759, is as follows:

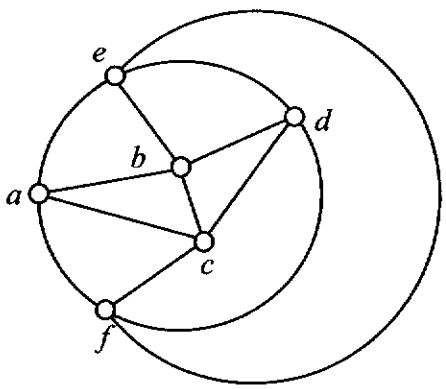


Figure 2.23. The final solution produced by the String Processing Algorithm.

The following table displays the total benefit of the final solutions produced by other algorithms using the same edge weights.

Algorithms	Total Benefit
1. Deltahedron Method (S)	751
2. Deltahedron Method (R)	732
3. Green-Al Hakim Algorithm	732
4. Leung's Constructive Heuristic	732
5. Wheel Expansion Method	761
6. Kim-Kim Algorithm	732
7. TESSA	754

Table 2.11. The total benefit of the final solutions.

The String Processing Algorithm has not been implemented on a computer, so there is no computational results available. Next we discuss some heuristic algorithms which are not in the group of graph theoretic based heuristics. We begin with simulated annealing.

2.3.2 Simulated Annealing

In local search, when we seek a solution and we are trapped in a possibly inferior region, there is no way to get out of this region and search for other solutions in different regions. The method of Simulated Annealing (SA), used to solve the facility layout design problem, starts with an initial solution and implements a local search to find an improved solution. SA can search other solutions in different regions, when a local search is trapped in a possibly inferior region, or when there is no way to get out of this region, by allowing the procedure to move to a better solution. In SA, if the new objective function value is better than that of the previous one, the new solution replaces the current solution based on the accepted difference between the objective function value of the previous solution and that of the new one. When the

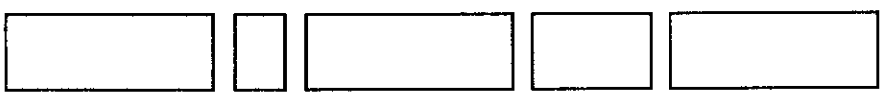
new solution has a value not better than the previous one, the new solution can be accepted based on a certain probability.

The description of SA technique, which is based on and derived from statistical mechanics, can be found in Johnson *et al.* (1989). SA has been explored successfully by Kirkpatrick *et al.* (1983), and Golden and Skiscim (1986). The application of SA for solving Quadratic Assignment Problems (QAP) was initiated by Burkard and Rendl (1984). Wilhelm and Ward (1987) used SA for solving the QAP. They used 1440 runs for the test problems in Nugent *et al.* (1968) and 144 runs for two test problems they constructed in their paper.

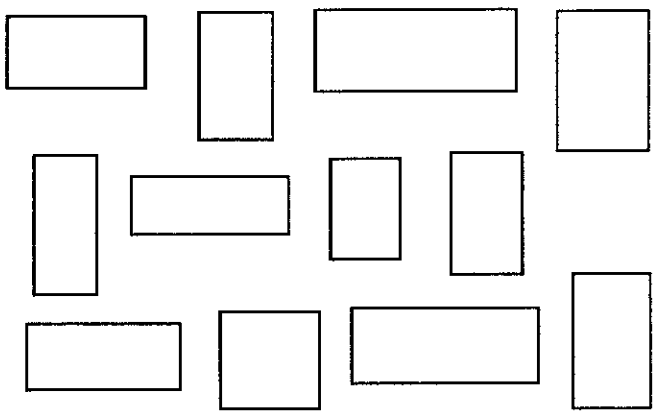
In recent years, there are a number of simulated annealing based algorithms for solving facility layout design problems. These include algorithms devised by Kouvelis and Kiran (1991), Heragu and Alfa (1992), Kouvelis *et al.* (1992), Kouvelis and Chiang (1992), Suresh and Sahu (1993), and Meller and Bozer (1996). A description of simulated annealing (theory and methodology) for facility layout design problem can also be found in Souilah (1995). In his description, the methodology of the general SA for manufacturing system layout design, is discussed along with its application in manufacturing cells, designing the intra-cell layout, and placing the manufacturing cells on the available shop-floor surface. Some numerical examples are presented without computational results.

Heragu and Alfa (1992) proposed SA algorithms for solving the facility layout design problem. They considered 2 or 3-facilities exchange between the positions of facilities. If the exchange between the position of the facilities results in a solution with a lower objective function value, then the exchange is made. This procedure is repeated until no two facilities can be exchanged without increasing the objective function value. They presented new models for the facility layout design problem and tested them for two patterns of layout frequently occurring in real world problems: single row, in which facilities are arranged linearly in one row; and multi-row, in which facilities are arranged linearly in two or more rows (see Figure 2.24). Their computational results for single-row problems show that for 8 test problems the first 6 problems produced optimal solutions. The other two are significantly better than those

in the literature. For multi-row layout problems, 15 problems were solved, including the first 8 small-sized problems (size 5 to 30) from Nugent *et al.* (1968) and 7 larger problems (size 42 to 90) from Skorin and Kapov (1990).



(a) Single-row layout.



(b) Multi-row layout.

Figure 2.24. Sample patterns for the facility layout problem.

Computational results for 8 single row layout problems (n = 5, 6, 7, 8, 12, 15, 20, 30) indicates that their algorithm has higher solutions than either 2 or 3 facility exchange algorithms. The output results of the SA algorithm of Wilhelm and Ward (1987) and that of Heragu and Alfa (1992), have the same objective function value for n=5,6,7,8, and 12. For n=15 and n=20, the SA algorithm of Heragu and Alfa (1992) are better than those of Wilhelm and Ward (1987). However for n=30, the SA algorithm of Heragu and Alfa (1992) indicates a better result than that of Wilhelm and Ward (1987).

Connolly (1990) proposed an improvement in SA procedure. From his testing, using 7 problems, he found that examining sequentially generated neighbouring

solutions, rather than randomly generated ones, is efficient for the implementation of SA.

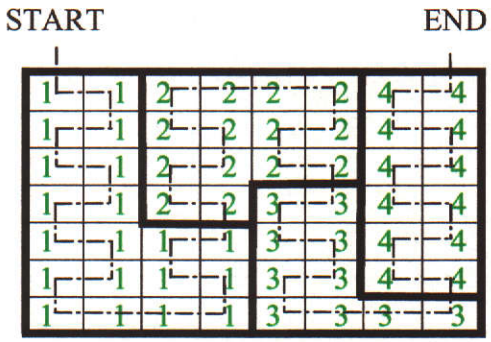
Kouvelis *et al.* (1992) proposed SA for machine layout problems in the presence of zoning constraints. Here a zoning constraint means a restriction on the arrangement of machines, which is often found in manufacturing environments. Positive zoning constraints require that machines are located next to each other, and negative zoning constraints force certain machines not to be close to each other. They proposed two distinct implementations of SA algorithm: the "Compulsion" and the "Penalty" methods for solving the facility layout problem.

The first algorithm, called "Compulsion method", is devoted to address the facility layout design problem, when there is the presence of the zoning constraints mostly during the search for a new layout in the neighborhood of the original configuration. The second algorithm, called the "Penalty method", is used in the presence of zoning constraints through the use of appropriate penalty terms in the objective function. They used a two-dimensional definition of the machine adjacency relationship, which we know as multi-row layout configurations. They assign the machines to the locations in a way that an appropriate cost function is minimized and the zoning constraints are not violated. The cost function consists of fixed costs, associated with locating a particular machine at a certain location; and costs associated with the flow per unit time between machines (for example, material-handling costs).

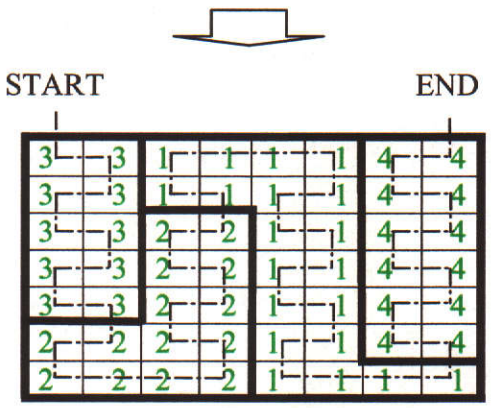
They conducted experiments using the test problems obtained from the classical study on QAP's by Nugent *et al.* (1968), Hillier (1963) and Golany and Rossenblatt (1989), with the problem size ranges from 5 to 30 machines. From their computational results, they found that the Compulsion Method outperforms the Penalty method both in terms of CPU time and total cost.

Meller and Bozer (1996) presented an application of simulated annealing to facility layout problems with single and multiple floors. They used a new candidate layout generation technique and "space filling curves" to develop an improvement-type layout algorithm based on simulated annealing that considers an expanded set of

department exchanges. The following figure depicts an illustration how spacefilling curves are used to determine the layout of a floor.



(a) Initial Layout with sequence:1-2-3-4.



(b) Department 1 and 3 are interchanged. Sequence: 3-2-1-4.

Figure 2.25. Using spacefilling curves to construct layouts.

In the above figure, the layout of the floor is determined by assigning each grid square to the departments of the layout sequence based on the path given by spacefilling curves. The number of grid squares assigned to a department is determined by the area of that department. A new layout is obtained by exchanging the departments in the layout sequence and reassigning each grid square to the appropriate department. Spacefilling curve generation and its impacts on the final layout are discussed in Bozer *et al.* (1994).

Meller and Bozer (1996) used three test problems which have 11, 15, and 25 departments. Their results on single-floor problems show that the new simulated annealing based algorithm, which they call "SABLE", outperforms "CRAFT" of Armour and Buffa (1963); "Probabilistic-search heuristic" of Nugent *et al.* (1968); "MULTIPLE" (Bozer *et al.*, 1994); Tam's simulated annealing (Tam, 1991); and "Pairwise exchange simulated annealing" algorithm presented by Wilhelm and Ward (1987).

2.3.3 Tabu Search

Another heuristic approach available for solving the facility layout design problems is Tabu Search (TS). The TS heuristic is a metaheuristic approach for solving combinatorial optimization problems. This approach can be superimposed on other algorithms, to avoid being trapped at a local optimal solution.

TS starts from an initial solution. To generate a list of candidate solutions, some local exchange heuristics are implemented. If there are many candidate solutions available, a restriction of search may be used to create a subset of the candidate solutions. Then, this subset of solutions is evaluated, followed by selecting the best solution from the candidate set of solutions, which has the maximum value (in maximization) or the minimum value (in minimization). The next best solution is selected if this solution has a forbidden status under a prescribed rule. This selected solution becomes the new current solution. To avoid the search going into an unwanted solution space, a tabu list is set up. This can avoid cycling.

To avoid giving a tabu status to an unvisited solution, aspiration level conditions are implemented. An exchange is said to satisfy an aspiration condition, if it helps the algorithm climb out of the local optima and search through other better solutions. This procedure is repeated until either the required number of iterations have been performed, or a given CPU time has been reached.

We can intensify the search in the region of solutions or explore another region. In order to intensify the search in the good regions, we should return to one of

the best solutions we have ever found. We can also conduct a partitioning of the solution region, by solving sub problems optimally. Combining all partial solutions will lead to an optimal solution. Intensification can also be carried based on long term memory. Here good moves and good solutions are memorized and evaluated. The best-visited solution which share a common feature, should be taken into account.

Diversification in tabu search is also used to avoid too many large unexplored neighbourhood areas in the search process. Diversification is conducted by performing several random different initial solutions (restarts).

The idea of Tabu Search (TS) was started in the 1970s, and was presented in its current form by Glover (1986) based on the basic ideas proposed by Hansen (1986). The formalization was conducted by Glover (1989), and Werra and Hertz (1989). The theoretical aspect of TS have been investigated by Faigle and Kern(1992), Glover (1992), and Fox(1993). Faigle and Kern (1992) have devised a probabilistic version of TS. In this technique, probability of moving can be chosen from a current solution to a neighbour in a very wide range, without losing the probabilistic convergence properties, can be done and is similar to refined simulated annealing procedure.

Tabu heuristic has been applied to solve the Quadratic Assignment Problem by Skorin-Kapov (1990). This technique, referred to as the tabu-navigation algorithm (T-N), is a simple tabu search, using a fixed tabu list and a simple neighbourhood scheme (neighbouring layouts are generated by exchanging the location of a pair of facilities). Neither intensification strategy, nor diversification strategy is applied in this method. Aspiration criteria to override the tabu status, and long term memory strategy for selecting new layouts are implemented for restarting the search. They found that the computational results indicate that the Tabu-navigation algorithm is better than Simulated Annealing, both in solution quality and computational times.

Skorin-Kapov (1991) modifies the Tabu-navigation program using a simple structure target analysis to guide the non-improving moves from the search algorithm. In this program intensification and diversification are also applied. To save on the

computational time, a restriction to neighbourhood search is applied. They also implement dynamic tabu list size strategies.

Taillard (1991) introduced parallelization methods to improve the speed of tabu search. He proposed a tabu search with the use of dynamic tabu list size, chosen randomly, to prevent the methods from cycling.

Hertz *et al.* (1997) proposed the description of basic ideas and some application of TS. They reformulate the classical steepest descent method, by adding aspiration level conditions, intensification strategy, and diversification strategy.

Chiang and Kouvelis (1996), implemented a tabu search heuristic, implementing long term memory structure, dynamic tabu list size strategies, intensification and diversification, and enhanced by the adoption of the sequential neighbourhood search. The size of dynamic tabu list is in between an upper and lower bound, in proportion to the percentage of improvement in terms of cost of the current move. In diversification the exploration step is spread over a different region of S, by implementing penalization efforts over different regions of S. This can penalize the non-improving moves, which have been tested in the previous iterations.

They tested the algorithm using extensive data of problem instances from Hillier (1963), Nugent *et al.* (1968), Krarup (n=30, unpublished data), Steinberg (1961), and Skorin-Kapov (1990). They also generated 60 problem instances with the size ranging from 50 through 100, using the test generator program of Li and Pardalos (1992). In their comparative analysis it was found, using problem instances from the above data, the algorithm from Chiang and Kouvelis (1996) yields the best solution and is time efficient. In terms of solution quality, their performance compared to the Tabu-Navigation method, using Skorin Kapov data, is about 4% lower than that of the Tabu Navigation method.

2.3.4 Expert System Method

In modeling the facility layout design, the quantitative objective function deals with minimizing material handling costs, which is related to the travel distance and the

handling system implemented. The qualitative objective function deals with some measure of the relationship chart or a closeness rating based on factors such as noise, supervision, or communication. The quantitative and qualitative objective can be integrated into the layout by involving factors not covered by the model. This idea is derived from Expert Systems (ES). The ES method has been applied to solve problems of manufacturing systems, particularly for job shop, assembly, assembly line, and flexible manufacturing as well as for solving the machine layout problem. Fisher and Nof (1984) used ES to solve general facility layout design problems by selecting equipment that satisfies the standard level and performing economic analysis. The predicate logics of knowledge based systems used in this method are procedures, facts, and goals, with flow, distance, and material handling cost as input information. A series of expert rules is used to evaluate the relativity chart.

Kumara *et al.* (1987) and Malakooti and Tsurushima (1989) proposed a heuristic-based ES for solving FLDP by formulating FLDP as a multi-objective problem with respect to the qualitative constraints. The number of departments (facilities) and their corresponding areas are used for the input information. They used knowledge base systems for drawing a square grid, drawing a screen and dividing it into equal areas, and then generating the adjacency. Further, Kumara *et al.* (1988) developed 2 different models using 3 assignment rules to determine the adjacency of 2 facilities. Leskowsky *et al.* (1987) selected the initial part-machine grouping, followed by laying out the machine areas. The input includes material flows, distance, and material handling costs. The machines are grouped into separate sets. The method constructs the layout of the group followed by constructing the total layout within each group. Kusiak and Heragu (1988) used Expert Systems in an automated manufacturing systems, using 35 rules, several different parameters, and evaluating the feasibility of the layout. The input covers the list of machines, the number of machines, the dimensions of machines, locations, type of layout and material handling system flow, closeness and relativity matrices. Abdou and Dutta (1990) integrated the combined effects of the selection of manufacturing system and material handling system to determine the suitable layout of machining facilities and to create the relativity chart.

They proposed the following factors to be considered in determining the configuration of the machines: product variety and quantity, degree of flexibility, level of automation, materials handling system, work-in-progress, environmental and safety considerations.

Sirinaovakul and Thajchayapong (1991,1994) stated that in some computerized techniques, the methods do not consider enough possible outcomes in the computer program to arrive at the optimal solution. Although there are a number of new modified algorithms presented to obtain a better solution, these methods are still very sensitive to the initial layout. The authors proposed 3 designed systems using a heuristic layout and heuristic search to solve the practical layout limitations systems: pattern generation algorithm, heuristic search and knowledge base (expert system). In this heuristic there are dual functions: heuristic function, which calculates the cost of layout and used as a criterion selection, and function system which selects a set of low-cost layouts. In calculating the cost, the expert system technique is used.

2.4 Discussions and Conclusions

The facility layout design problem has been addressed over many years using a variety of methods, including exact and heuristic approaches. Several mathematical models have also been devised in dealing with the problem. To date, there are more heuristics algorithms than exact algorithms. In the recent years, constructive graph theoretic based heuristics have become popular.

In nearly all graph theoretic based heuristics, the insertion operation maintains the planarity of the generated graph. The resulting partial subgraph, as a partial solution, is always maximal planar. TESSA is the only graph theoretic heuristic that does not maintain maximal planarity of its partial subgraphs, although it starts with a maximal planar graph K_3 as its initial solution and ends up with a maximal planar graph as its final solution.

In TESSA the process of construction employs a face addition to the boundary. This type of insertion or addition can create a new vertex of degree 3, 4, or even more.

Thus, TESSA can construct any type of Maximal Planar Graph (MPG), including an MPG which has minimum degree of four or five. Boswell (1992) stated that this type of graph cannot be obtained by using The Deltahedron Method.

In TESSA, inserting a face to the interior is not permitted. It is only added to the boundary. However, there are some high-weighted faces, which cannot be inserted just because they do not fit to the edge (edges) on the boundary. This means in some instances good solutions may not be obtained. Some high-weighted faces may be rejected from the insertion, when they have 1 or 2 edges in the interior, or have no common edges with the boundary. There are also some faces which are rejected several times before they are inserted into the existing subgraph. All these advantages make the algorithm time consuming.

Kim and Kim (1995) stated that the Deltahedron Method and Wheel Expansion exhibit an umbrella effect, a state where one facility is adjacent to all the others. This effect may yield a complicated process in converting a graph into a block layout (Giffin *et al.* 1986). To solve this problem Kim and Kim (1995) introduced an algorithm which offers 3-vertex insertion in each iteration. In their algorithm, the 3-vertex insertion operation is always carried out until 2 or 1 vertex is left and 1-vertex insertion is implemented only at the end. This procedure is done to avoid an "umbrella effect" and to save computing time. However, this procedure may not produce additional benefits in each insertion operation.

Most of the known graph theoretic based heuristics for solving the facilities layout design problem implement a construction procedure, which is started by constructing a triangle or a tetrahedron as an initial solution. The process is started by selecting, from a set of 3 or 4 vertices, a triangle or a tetrahedron with the highest weight. The solution is then grown iteratively by inserting 1 vertex or more into the existing partial solution, using a well defined insertion operation which maintains the planarity of the generated graph. This operation is carried out repeatedly until all vertices have been inserted and a maximal planar graph consisting of $3n$-6 edges is obtained.

The drawback of these methods (which do not implement improvement procedures), however, is that there may be some edges, that contribute poorly to the objective function value and so a poor-weighted solution may be generated. This means we need a time-efficient technique which can alleviate this problem, by removing a poor-weighted edge from a partial solution throughout the insertion process. Further, due to the limitation of the improvement techniques applied, the resulting configuration of final graphs may not be improved at all. For example, in the vertex relocation (the vertex movement) technique, only a vertex of degree 3 can be relocated to a new face in the existing graph, so there is no improvement can be made for a graph containing all vertices with degree more than 3. Often, most subgraphs or subdivisions of the existing final graph (final solution) are not suitable to the conversion specified by the improvement technique used, so no additional benefit can be obtained. This means that some poor-weighted edges may still be in the resulting final solution.

In the next Chapter we will present 3 graph theoretic based heuristics which are capable of constructing any type of MPG, including an MPG which has minimum degree of four or five. These heuristics can evaluate all edges in the current partial subgraph and may remove them to improve the weight of the graph.

CHAPTER 3

NEW GRAPH THEORETIC HEURISTICS

As we noted in the previous Chapter, the available literature graph theoretic heuristics are greedy construction procedures. Once a vertex and the relevant edges are added, they remain in the solution. This means that some poor-weighted edges may be present in the final solution generated. The new heuristics that we develop in this Chapter address this problem by allowing, at each stage, edges to be removed from the partial solution if this improves the objective function value.

In this Chapter we present 3 new graph theoretic based heuristics along with their performance, compared to the literature heuristics. Our computational results are based on 4200 randomly (uniform and normal distribution) generated problems with n, the number of facilities, ranging from 5 to 100.

3.1 New Graph Theoretic Algorithms

In all new heuristics, the initial solution is the highest-weighted complete subgraph K_4. This is selected from a list of all possible subgraphs consisting of 4 vertices. The insertion procedure in our heuristics allows for edge-removal and multiple vertex insertions, including 1-vertex and 2-vertex insertions. This option can produce a higher-weighted partial subgraph with all vertices having degree greater than 3. This includes octahedron and icosahedron, which cannot be produced by using the construction technique of the Deltahedron Method. So, our heuristic does not restrict the type of Maximal Planar Graph (MPG) produced.

Vertices from the set of unused vertices are inserted into the existing partial solution until they are all assigned. We insert 1 or 2 vertices at a time to a face or a cycle of order 4 in the current partial solution. By this insertion technique we may drop a poor-weighted edge in the existing partial solution, and replace it with a better-weighted edge. An insertion of 1 vertex or 2 vertices to a cycle of order 4 drops an edge, particularly a poor-weighted edge, which does not give a good contribution to the current partial solution.

Since a triangulation process is used at each step, the maximal planarity is always maintained in each stage. Thus our algorithms do not need a planarity testing routine. In each iteration, we always insert 1 or 2 vertices to the existing partial subgraph. So, our heuristic are greedy and thus time efficient.

We begin our discussion with some simple insertion operations. The basic operations used replace a subgraph H with a subgraph H'. These operations do not produce a new edge outside of the subgraph H, so the resulting subgraph will not contain a braced edge. We now describe 6 insertion operations. In our illustrations, we adopt the convention that new vertices are not shaded and the new edges are not bold.

We start with a single vertex insertion operation O_1, displayed in Figure 3.1. Here, a new vertex is inserted into the existing face, producing a new vertex of degree

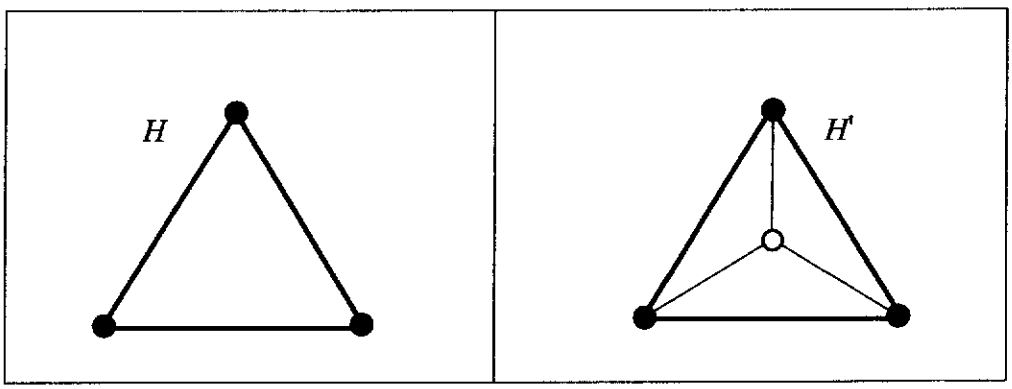


Figure 3.1. Insertion Operation O_1.

three, three new edges, and three new faces. The existing face is eliminated from the partial subgraph.

Now we consider another single vertex insertion. Let H be a subgraph consisting of two faces which has a common edge as displayed in Figure 3.2. Operation O_2 removes the common edge from subgraph H to obtain a cycle of order 4 and inserts a new vertex into the cycle and joins it to all four vertices of H to obtain the subgraph H'. Here subgraph H' has four new edges and four new faces. Two faces are removed from the current partial subgraph. This operation has an advantage, as it evaluates the weight of each edge in the current partial solution. In this operation, an edge, which is a common edge of two faces, can be removed. This process can overcome the problem existing in most of the established graph theoretic based heuristics, where poor-weighted edges maybe accumulated in each partial solution, and thus in the final solution.

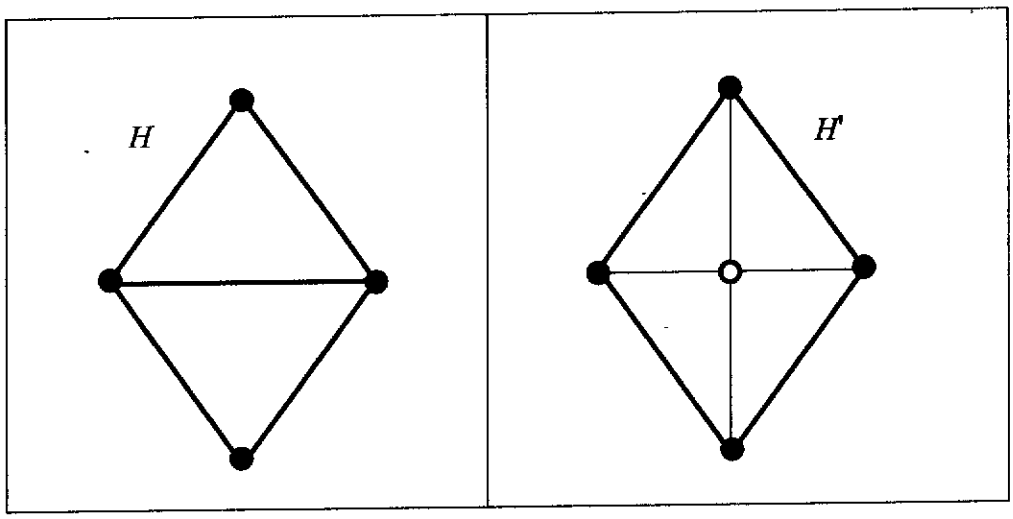


Figure 3.2. Insertion Operation O_2.

Now we focus our attention to a double vertex insertion operation: Operation O_3. As illustrated in Figure 3.3, we implement a double vertex insertion into a triangular face H and get the subgraph H' which contains 5 new faces and 6 new edges. One face is removed from the current partial subgraph. Note that there are 3 different configurations to evaluate depending on which of the three vertices

of H is not joined to the new vertex of degree 3. This insertion operation does not eliminate an edge from the current partial subgraph.

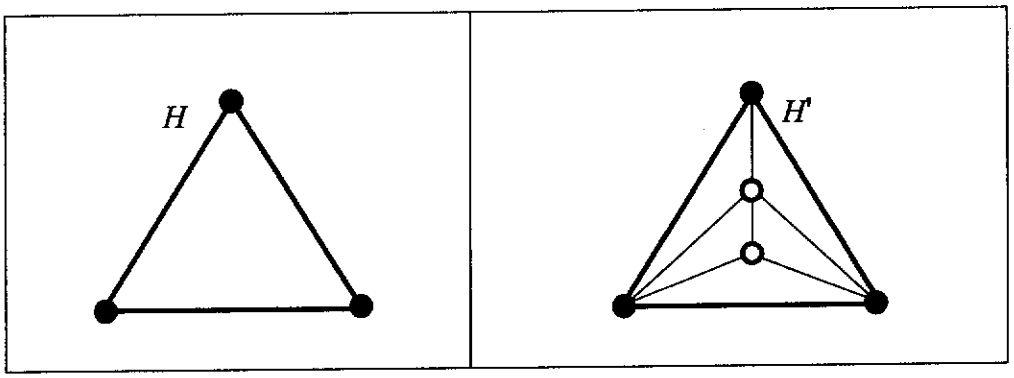


Figure 3.3. Insertion Operation O_3.

Next we consider the second double vertex insertion: Operations O_4. Firstly, by removing the common edge of two faces in a subgraph H, and inserting 2 new vertices such that both have degree 4, we end up with subgraph H', as illustrated in Figure 3.4. Here subgraph H' has 7 new edges, and 6 new faces. The previous two faces are removed. Note that in each H' there are 2 possible different configurations which have to be considered in the weight computation. Although both new

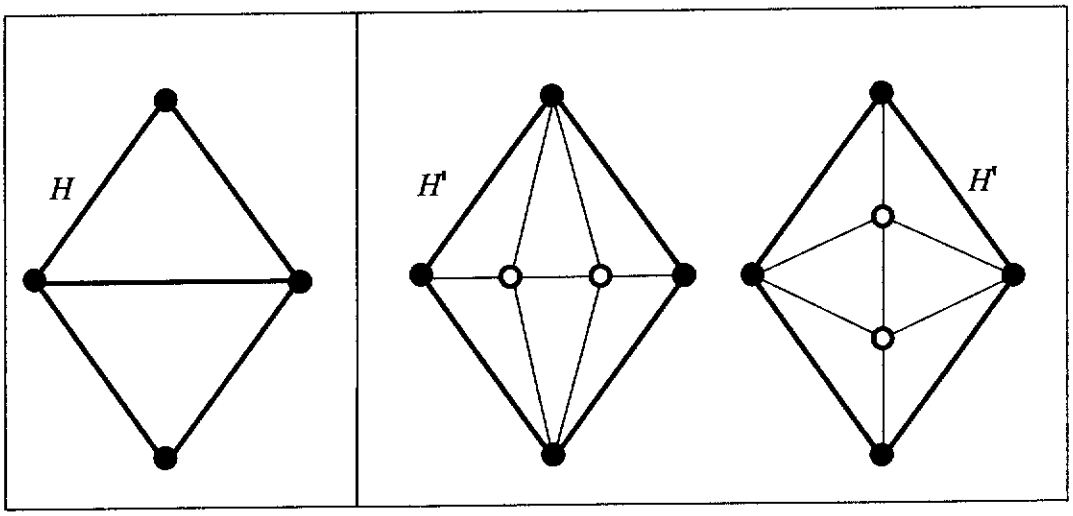


Figure 3.4. Insertion Operation O_4.

vertices are of degree 4, the vertices adjacent to these two vertices are different. Again, in this type of insertion operation we have an opportunity to eliminate an edge which may contribute a poor weight to the existing partial subgraph.

Next we consider the third double vertex insertion operation: Operation O_5. By removing the common edge from the subgraph H, and inserting 2 new vertices, one with degree 3 and the other with degree 5, we have the subgraph H' which has 7 new edges and 6 new faces (Figure 3.5). We also remove 1 existing edge and 2 existing faces from the current partial subgraph. Note that there are four possible configurations of H' obtained by fixing the "inner part" and rotating clockwise the outer vertices that need to be evaluated. Using this type of insertion operation we again may drop a poor-weighted edge from the existing partial subgraph.

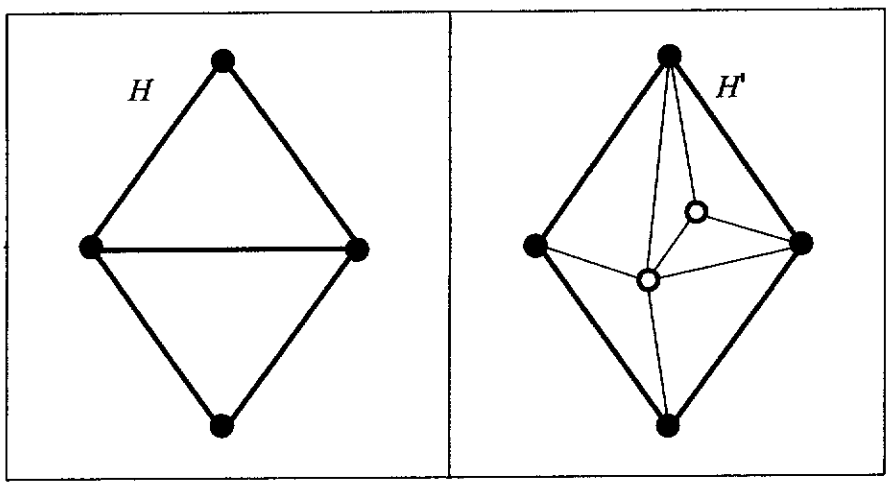


Figure 3.5. Insertion Operation O_5.

Now we consider a triple vertex insertion operation, which was suggested by Leung (1992). In this operation (Operation O_6), where three vertices are inserted into a face, there are 6 different possible configurations of subgraph H'. One of them is illustrated in Figure 3.6. The other possible configurations are displayed in Chapter 2, Figure 2.12.

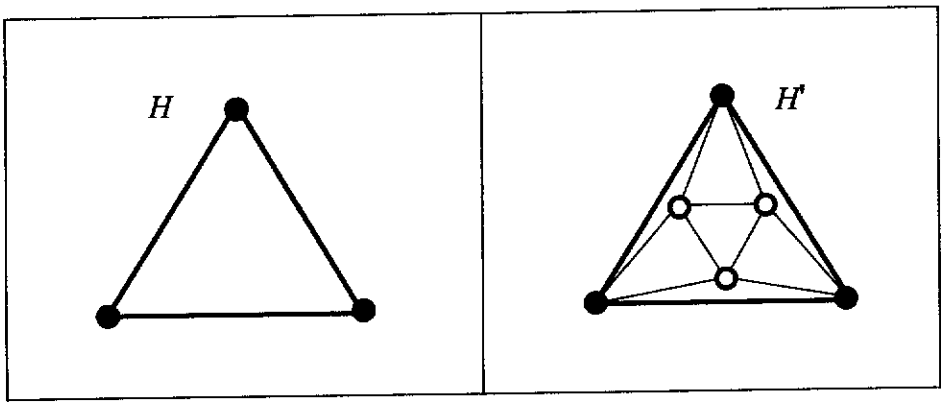


Figure 3.6. Insertion Operation O_6.

By using the above operations we can describe 3 new heuristics CK-1, CK-2, and CK-3 as follows.

Algorithm CK-1:

Step 1. Initialization

Select best K_4.

Step 2. Generation of candidates

Consider all the operations O_1, O_2, and O_3, on the current partial subgraph.

Step 3. Construction

Choose the highest-weight insertion per added vertex and construct a new partial subgraph based on the corresponding type of insertion. Repeat until all unused vertices are inserted.

Algorithm CK-2 (Cacccetta and Kusumah, 2000) can be considered as the extension of Algorithm CK-1 (Cacccetta and Kusumah, 1998). The process of obtaining a K_4 as its initial solution is exactly the same as that in CK-1. In Algorithm CK-2, there are 3 types of 1-vertex insertions and 7 types of 2-vertex insertions. We consider 2-vertex insertions which give 2 resulting vertices of degree 3 and 5 respectively. From the computational results presented in the next

section we can see that the additional features in CK-2 lead to an improved final solution. The details of the algorithm is as follows:

Algorithm CK-2:

Step 1. Initialization

Select best K_4.

Step 2. Generation of candidates

Consider all the operations O_1, O_2, O_3, and O_5 on the current partial subgraph.

Step 3. Construction

Choose the highest-weight insertion per added vertex and construct a new partial subgraph based on the corresponding type of insertion. Repeat until all unused vertices are inserted.

We also explore the insertion of 2 vertices at the same time into a face (Caccetta and Kusumah, 1999), so that 2 new resulting vertices are of degree 3 and 4, respectively. This additional type of insertion, which is incorporated in Algorithm CK-3, has increased the performance of Algorithm CK-2. The detailed construction of algorithm CK-3 is as follows.

Algorithm CK-3:

Step 1. Initialization

Select best K_4.

Step 2. Generation of candidates

Consider all the operations O_1, O_2, O_3, O_4, and O_5 on the current partial subgraph.

Step 3. Construction

Choose the highest-weight insertion per added vertex and construct a new partial subgraph based on the corresponding type of insertion. Repeat until all unused vertices are inserted.

In terms of the above operations (Operations O_1 to O_6) we can conveniently describe the main constructive graph theoretic based heuristics (except for the Wheel Expansion Method, TESSA, and the String Processing Algorithm) as follows:

Algorithms	Insertion Methods	Complexity
1. Deltahedron Method – (S) (Foulds and Robinson, 1978)	O_1	$O(n^2)$
2. Deltahedron Method – (R) (Foulds and Robinson, 1978)	O_1	$O(n^4)$
3. Green-Al Hakim Algorithm (Green and Al Hakim, 1985)	O_1	$O(n^3)$
4. Leung's Constructive Heuristic (Leung, 1992)	O_1, O_6	$O(n^5)$
5. Kim-Kim Algorithm (Kim and Kim, 1985)	O_1, O_6	$O(n^5)$
6. New Algorithm CK-1	O_1, O_2, O_3	$O(n^4)$
7. New Algorithm CK-2	O_1, O_2, O_3, O_5	$O(n^4)$
8. New Algorithm CK-3	O_1, O_2, O_3, O_4, O_5	$O(n^4)$

Table 3.1. Constructive graph theoretic based heuristics and their insertion operations.

3.2. Computational Results

In order to evaluate the ten heuristics described in Chapter 2, Section 2.3, and Chapter 3, Section 3.1, a comparative analysis has been carried out based on 4200 randomly generated problems. We now discuss the details of the test-problem size, generation of random test problems, and implementation. The size of the test problems is expressed by the number n of vertices (facilities), where n ranged from 5 to 100 in increments of 5. For each n, 30 test problems were generated by selecting the edge weights (benefit) from a uniform distribution [0,100]. This gives rise to 600 problems. Again, for each n, another 30 test problems were generated by selecting the edge weight (benefit) from a normal distribution with mean 100 and standard deviation ranging from 5 to 30 in increments of 5. This gives rise to 3600 problems. The computational work was carried out on a Silicon Graphic Workstation (R5000) running at clock speed of 150 MHz. All algorithms were implemented in C.

Table 3.2 and Figure 3.7 display the performance of each algorithm compared to the upper bound. In both presentations, the edge weights used are randomly generated from uniform distribution [0,100]. In Tables 3.3-3.8 and Figures 3.8 to 3.13, the edge weights used are randomly generated from normal distribution with standard deviations 5, 10, 15, 20, 25, and 30. In each cell of these tables, the displayed value depicts the average weight of 30 final solutions (as a percentage of the upper bound) for each associated size of vertices. It should be noted that the value of the upper bound, given by the total weight of the best possible $3n$-6 edges, is not necessarily the optimal value, since the representing graph may not be planar.

From these tables and figures, we can see that the best performing heuristic is Algorithm CK-3, and the lowest one is TESSA. For $n \leq 50$, the Wheel Expansion Method shows a significantly better performance (for edge weights from normal distribution) than that of the Leung's. However for $n > 50$ the Leung's always provide better solutions than those provided by the Wheel Expansion Method.

The computational times (CPU times) of the heuristics are given in Tables 3.9 to 3.10, and charts in Figures 3.14 to 3.15. Note that for the edge weights generated from randomly normal distribution, we only present the computational times for standard deviation 5, as the computational times of the other standard deviations are the same. From these tables and charts we can see that the most efficient heuristic, in terms of time, is the Deltahedron Method (S-Construction). The main aspect to note is that the solution generated by TESSA needs a huge amount of computing time. As can be seen from Figure 3.14 and Figure 3.15, for $n \leq 35$, almost all algorithms, including TESSA, have the same CPU time. However, TESSA shows a dramatic increase of CPU time for $n>50$. For $n=75$, for example, it spends almost 800 seconds for solving one problem, and it needs more than 4,500 seconds for a problem of size $n=100$ (see Figure 3.14).

In the Kim-Kim algorithm, the triple vertex insertion operation is always carried out until 1 vertex or two vertices are left, and a single vertex insertion is implemented only in the last iterations. This procedure apparently can save time compared to the procedure in the Leung's Constructive Heuristic; as can be observed either from Table 3.9 or Table 3.10. However, as indicated in Table 3.2 to 3.8, there is no advantage of selecting the best solution obtained from a single vertex insertion.

In terms of solution quality, the performance of the above algorithms can be presented in rank (from the highest to the lowest) as: Algorithm CK-3, Algorithm CK-2, Algorithm CK-1, the Wheel Expansion Method, the Deltahedron Method (R-Construction), the Green-Al Hakim Algorithm, the Kim-Kim Algorithm, the Deltahedron Method (S-Construction), TESSA. Based on their CPU time, the rank of their performance (from the shortest to the longest) is as follows: the Deltahedron Method (S-Construction), Algorithm CK-1, Algorithm CK-2, the Green-Al Hakim Algorithm, the Deltahedron Method (R-Construction), Algorithm CK-3, the Leung's Constructive Heuristic, the Wheel Expansion Method, TESSA.

As can be seen from Figures 3.7 to 3.13, the results presented for different group of problems (from uniform distribution and from normal distribution with different standard deviations) may be different, but they always show the same

general tendency. The rank of each algorithm, in terms of its average total benefit, is the same for all data with different standard deviation. However, the higher the standard deviation, the lower the performance.

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.89	99.89	99.62	99.89	100.00	97.44	99.89	100.00	100.00	100.00
10	92.41	92.62	92.30	93.72	94.29	91.46	89.53	94.48	94.62	95.15
15	89.84	90.67	90.77	91.63	91.85	89.20	90.21	91.75	92.21	92.62
20	87.93	89.88	89.82	90.98	90.69	87.40	88.28	91.54	91.38	91.69
25	87.40	89.77	89.86	90.65	90.28	87.76	87.35	90.60	90.91	91.35
30	87.32	89.72	89.55	90.66	90.43	87.64	88.60	90.93	91.11	91.45
35	87.30	89.63	89.48	90.72	90.13	87.17	88.46	90.84	90.96	91.46
40	87.40	89.60	89.49	90.71	90.28	86.79	88.15	90.89	90.91	91.28
45	87.28	90.01	89.91	90.77	90.51	86.78	88.84	90.97	91.08	91.35
50	87.09	90.05	89.99	90.69	90.35	87.56	88.87	90.92	91.16	91.66
55	87.41	90.12	89.88	90.98	90.51	86.92	88.78	91.10	91.25	91.57
60	87.42	90.49	90.28	91.24	90.69	87.05	89.34	91.06	91.33	91.75
65	87.49	90.28	90.37	91.32	90.76	86.71	89.22	91.25	91.30	91.86
70	87.63	90.56	90.53	91.50	90.83	87.42	89.19	91.50	91.61	91.99
75	87.96	90.60	90.75	91.57	91.06	87.10	89.99	91.60	91.68	92.07
80	87.75	90.78	90.78	91.58	91.08	87.07	89.79	91.44	91.65	92.03
85	87.87	90.84	90.98	91.79	91.21	87.67	89.64	91.83	91.97	92.27
90	87.96	91.03	90.93	91.86	91.37	87.85	90.13	91.75	91.91	92.37
95	88.28	91.15	91.15	92.04	91.34	87.14	90.15	92.01	92.15	92.54
100	88.12	91.24	91.20	92.03	91.44	87.74	90.10	92.04	92.26	92.43

Table 3.2. The Average Total Benefit (% of Upper Bound, Uniform Distribution [0,100]).

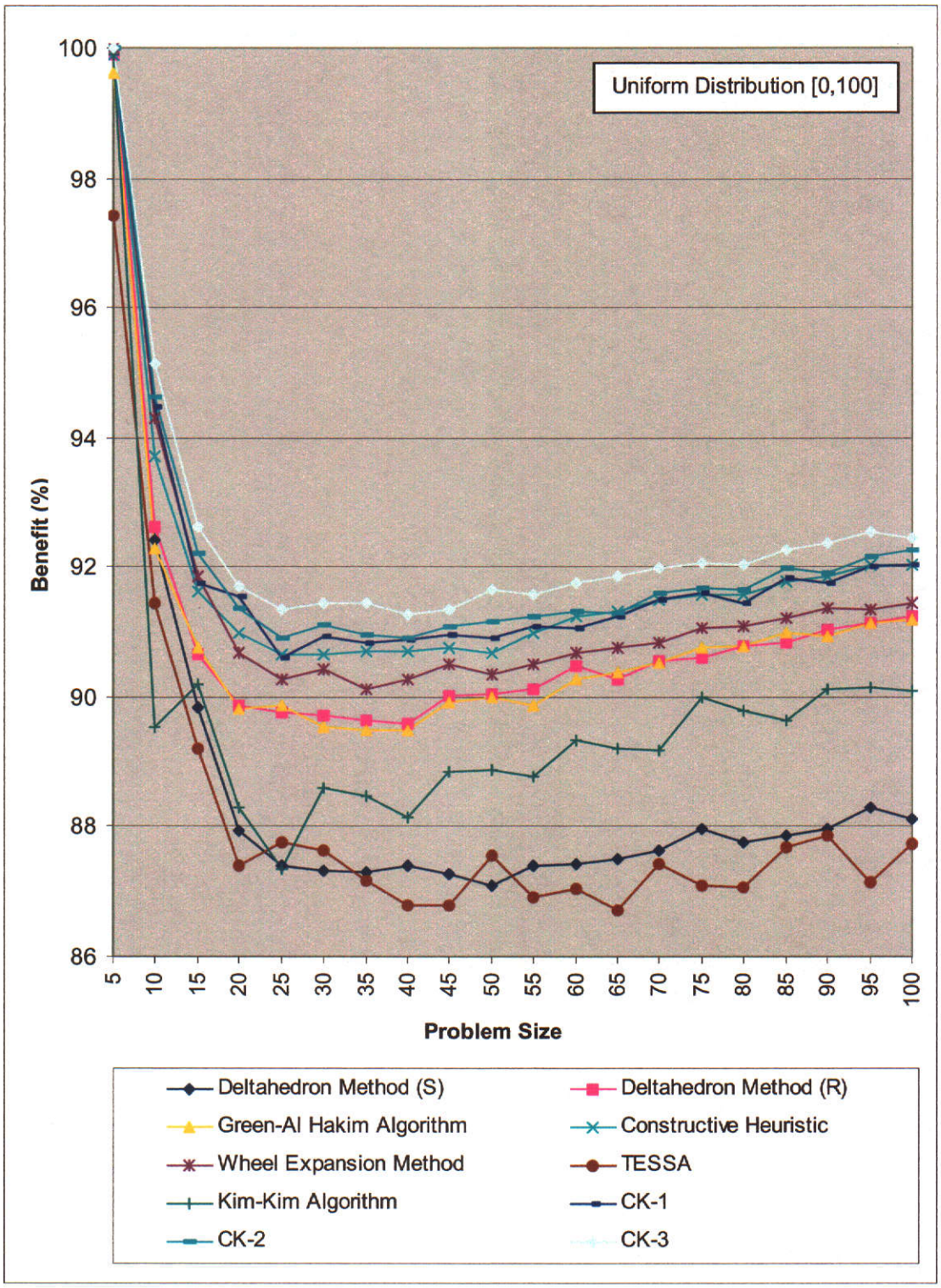


Figure 3.7. The Average Total Benefit (% of Upper Bound).

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Constructive Heuristic	Wheel Expansion	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.95	99.95	99.92	99.95	100.00	99.75	99.95	100.00	100.00	100.00
10	99.31	99.35	99.31	99.41	99.52	99.05	98.95	99.51	99.51	99.56
15	98.86	98.91	98.91	99.02	99.06	98.68	98.82	99.10	99.13	99.13
20	98.43	98.56	98.58	98.69	98.75	98.31	98.39	98.83	98.86	98.87
25	98.24	98.38	98.40	98.57	98.59	98.16	98.16	98.68	98.69	98.76
30	98.02	98.26	98.26	98.40	98.44	97.90	98.17	98.51	98.50	98.57
35	97.88	98.14	98.16	98.33	98.34	97.81	98.00	98.42	98.46	98.51
40	97.70	98.03	98.00	98.18	98.22	97.60	97.75	98.27	98.31	98.35
45	97.59	97.90	97.90	98.15	98.11	97.63	97.86	98.22	98.24	98.26
50	97.53	97.85	97.86	98.06	98.05	97.46	97.76	98.13	98.17	98.19
55	97.42	97.80	97.81	97.99	98.02	97.39	97.60	98.08	98.11	98.14
60	97.38	97.75	97.76	97.94	97.95	97.27	97.67	98.00	98.04	98.06
65	97.29	97.70	97.70	97.88	97.88	97.11	97.54	97.93	97.97	98.06
70	97.22	97.63	97.62	97.84	97.82	97.11	97.43	97.89	97.93	97.96
75	97.15	97.58	97.57	97.78	97.76	97.09	97.51	97.87	97.89	97.93
80	97.18	97.53	97.53	97.76	97.79	97.03	97.48	97.83	97.84	97.92
85	97.07	97.50	97.51	97.69	97.68	96.96	97.34	97.74	97.80	97.83
90	97.03	97.44	97.44	97.68	97.63	96.90	97.42	97.73	97.76	97.82
95	97.04	97.44	97.44	98.30	97.61	96.85	97.33	97.72	97.74	97.80
100	96.93	97.34	97.37	97.60	97.54	96.84	97.25	97.63	97.68	97.73

Table 3.3. The Average Total Benefit (% of Upper Bound, Normal Distribution with Mean=100 and SD=5).

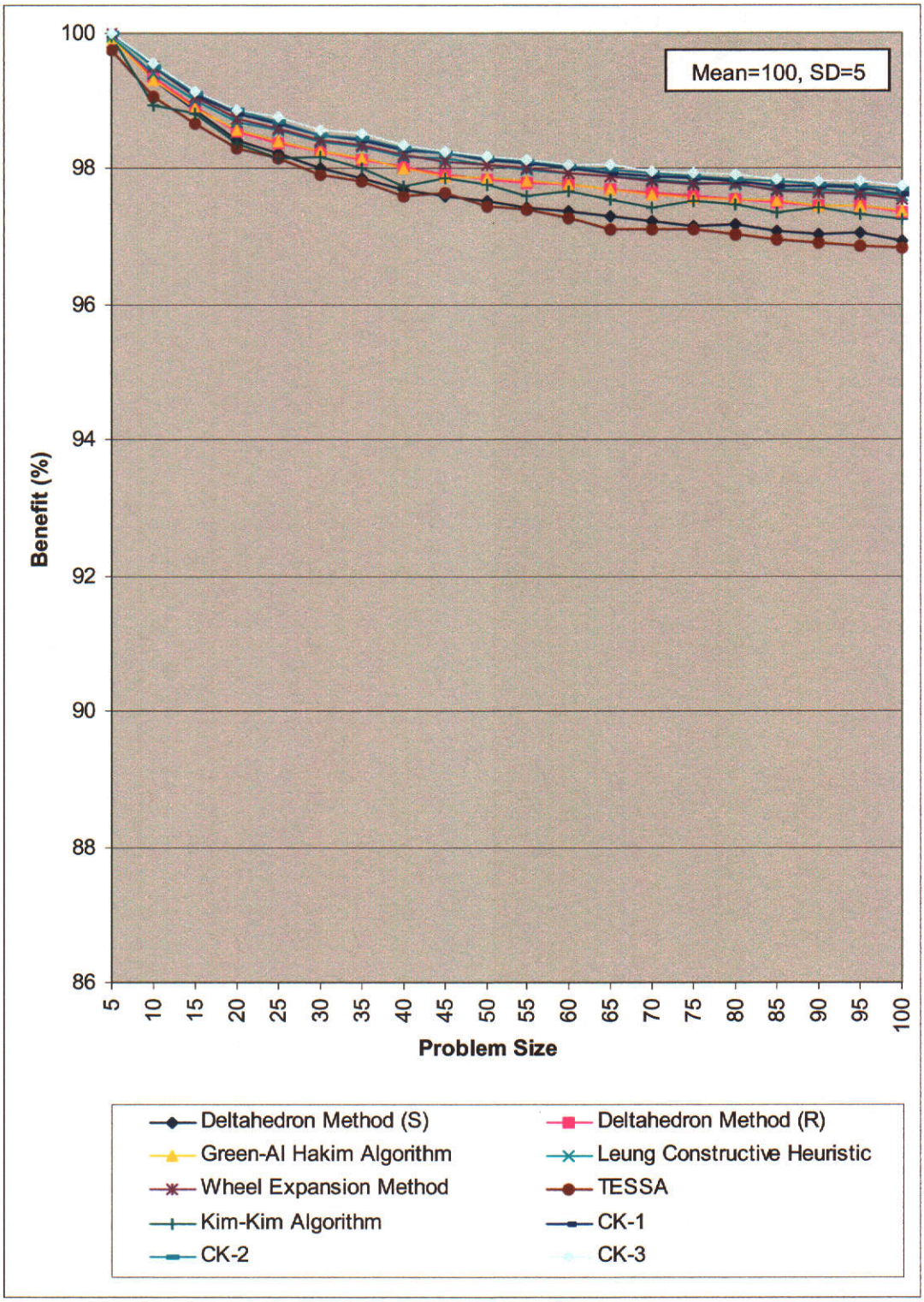


Figure 3.8. The Average Total Benefit (% of Upper Bound, Mean=100, SD=5).

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.90	99.90	99.83	99.90	100.00	99.50	99.90	100.00	100.00	100.00
10	98.63	98.81	98.75	98.90	99.07	98.23	97.90	99.03	99.04	99.19
15	97.82	97.89	97.87	98.11	98.26	97.37	97.78	98.22	98.29	98.39
20	96.97	97.15	97.21	97.55	97.67	96.60	96.93	97.79	97.80	97.86
25	96.63	96.97	97.00	97.28	97.47	96.34	96.54	97.48	97.50	97.63
30	96.33	96.81	96.78	96.99	97.06	95.99	96.59	97.20	97.22	97.35
35	96.04	96.53	96.50	96.88	96.89	95.93	96.29	97.10	97.10	97.18
40	95.71	96.31	96.28	96.66	96.72	95.51	95.89	96.81	96.85	96.98
45	95.54	96.10	96.06	96.50	96.49	95.43	96.00	96.71	96.75	96.79
50	95.43	96.02	96.02	96.36	96.36	95.21	95.86	96.51	96.62	96.66
55	95.21	95.96	95.92	96.29	96.30	95.08	95.61	96.51	96.51	96.60
60	95.14	95.87	95.80	96.17	96.21	94.93	95.65	96.37	96.40	96.44
65	94.98	95.69	95.71	96.03	96.00	94.83	95.47	96.21	96.31	96.38
70	94.84	95.61	95.58	95.99	95.93	94.58	95.31	96.18	96.19	96.29
75	94.84	95.51	95.48	95.89	95.91	94.63	95.43	96.13	96.14	96.20
80	94.72	95.49	95.43	95.90	95.83	94.56	95.31	96.04	96.06	96.18
85	94.56	95.36	95.38	95.75	95.72	94.38	95.14	95.95	95.93	96.10
90	94.51	95.30	95.28	95.68	95.64	94.27	95.26	95.88	95.93	95.97
95	94.50	95.20	95.23	95.67	95.61	94.15	95.10	95.77	95.83	95.93
100	94.35	95.15	95.17	95.61	95.54	94.04	94.99	95.69	95.79	95.88

Table 3.4. The Average Total Benefit (% of Upper Bound, Normal Distribution with Mean=100, SD=10).

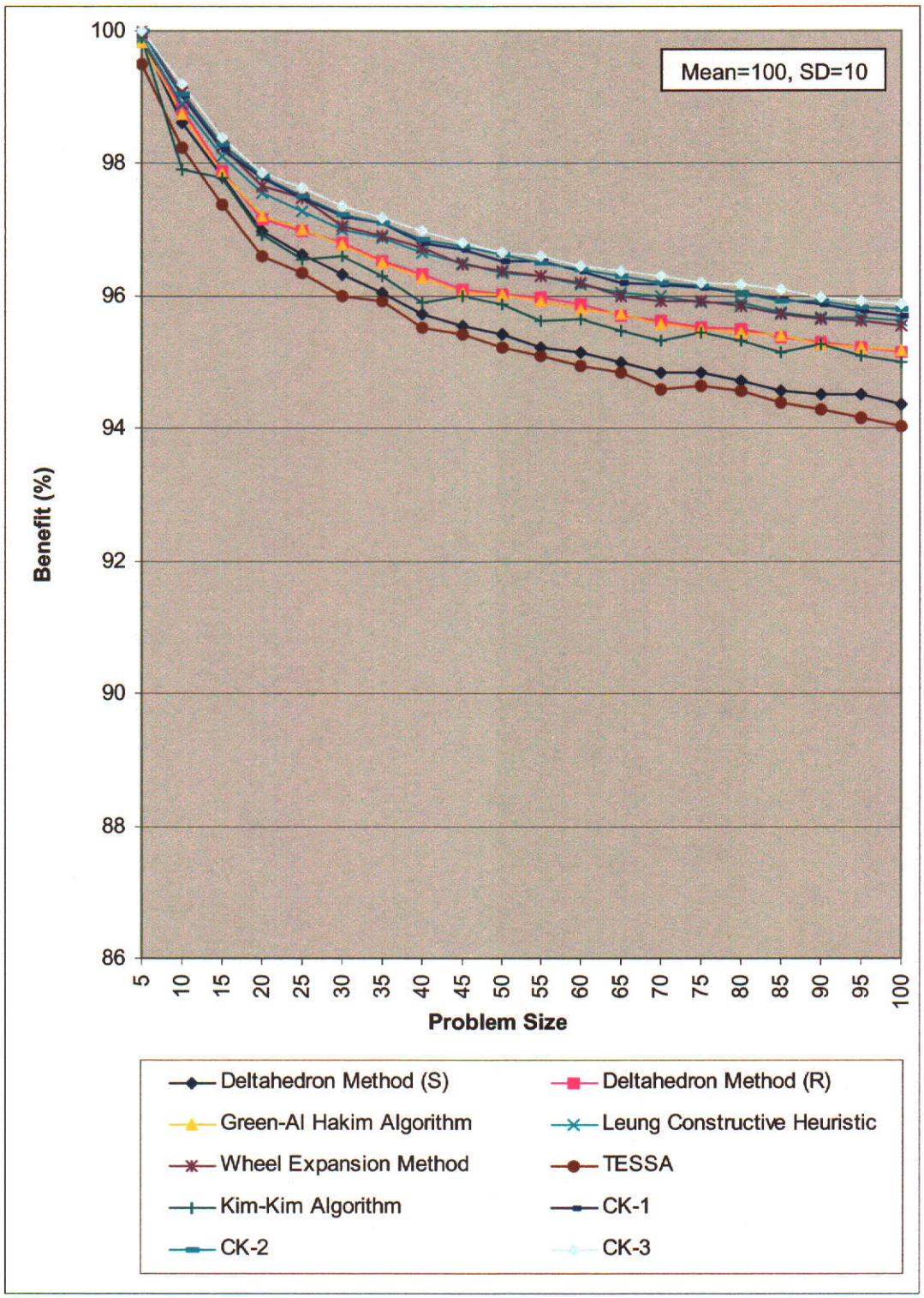


Figure 3.9. The Average Total Benefit (% of Upper Bound, Mean=100, SD=10).

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.84	99.84	99.73	99.84	100.00	99.24	99.84	100.00	100.00	100.00
10	98.05	98.30	98.13	98.34	98.66	97.27	96.91	98.58	98.63	98.82
15	96.90	97.00	96.97	97.29	97.53	96.17	96.66	97.44	97.54	97.70
20	95.65	96.02	96.15	96.39	96.67	95.26	95.60	96.87	96.91	97.01
25	95.15	95.70	95.69	96.15	96.40	94.95	95.10	96.49	96.48	96.71
30	94.88	95.51	95.54	95.75	95.96	94.60	95.10	96.07	96.18	96.24
35	94.42	95.20	95.05	95.60	95.73	94.20	94.87	95.92	95.93	96.08
40	94.01	94.78	94.75	95.29	95.32	93.77	94.13	95.51	95.63	95.70
45	93.75	94.57	94.54	95.06	95.09	93.36	94.46	95.37	95.41	95.55
50	93.54	94.40	94.35	94.91	94.93	93.30	94.21	95.14	95.23	95.34
55	93.32	94.33	94.37	94.86	94.86	93.15	93.85	95.11	95.07	95.26
60	93.20	94.24	94.23	94.65	94.65	92.83	94.00	94.87	94.91	95.05
65	93.00	94.05	94.01	94.52	94.43	92.70	93.72	94.73	94.82	94.97
70	92.79	93.90	93.86	94.43	94.39	92.44	93.44	94.61	94.67	94.83
75	92.87	93.75	93.76	94.34	94.34	92.59	93.67	94.56	94.57	94.76
80	92.69	93.76	93.68	94.25	94.29	92.40	93.51	94.46	94.55	94.70
85	92.42	93.54	93.63	94.06	94.03	92.22	93.21	94.36	94.34	94.55
90	92.39	93.42	93.42	94.01	93.96	92.11	93.39	94.19	94.28	94.42
95	92.35	93.43	93.40	93.96	93.90	92.04	93.22	94.15	94.18	94.38
100	92.15	93.31	93.29	93.92	93.78	91.92	93.07	93.99	94.17	94.25

Table 3.5. The Average Total Benefit (% of Upper Bound, Normal Distribution with Mean=100, SD=15).

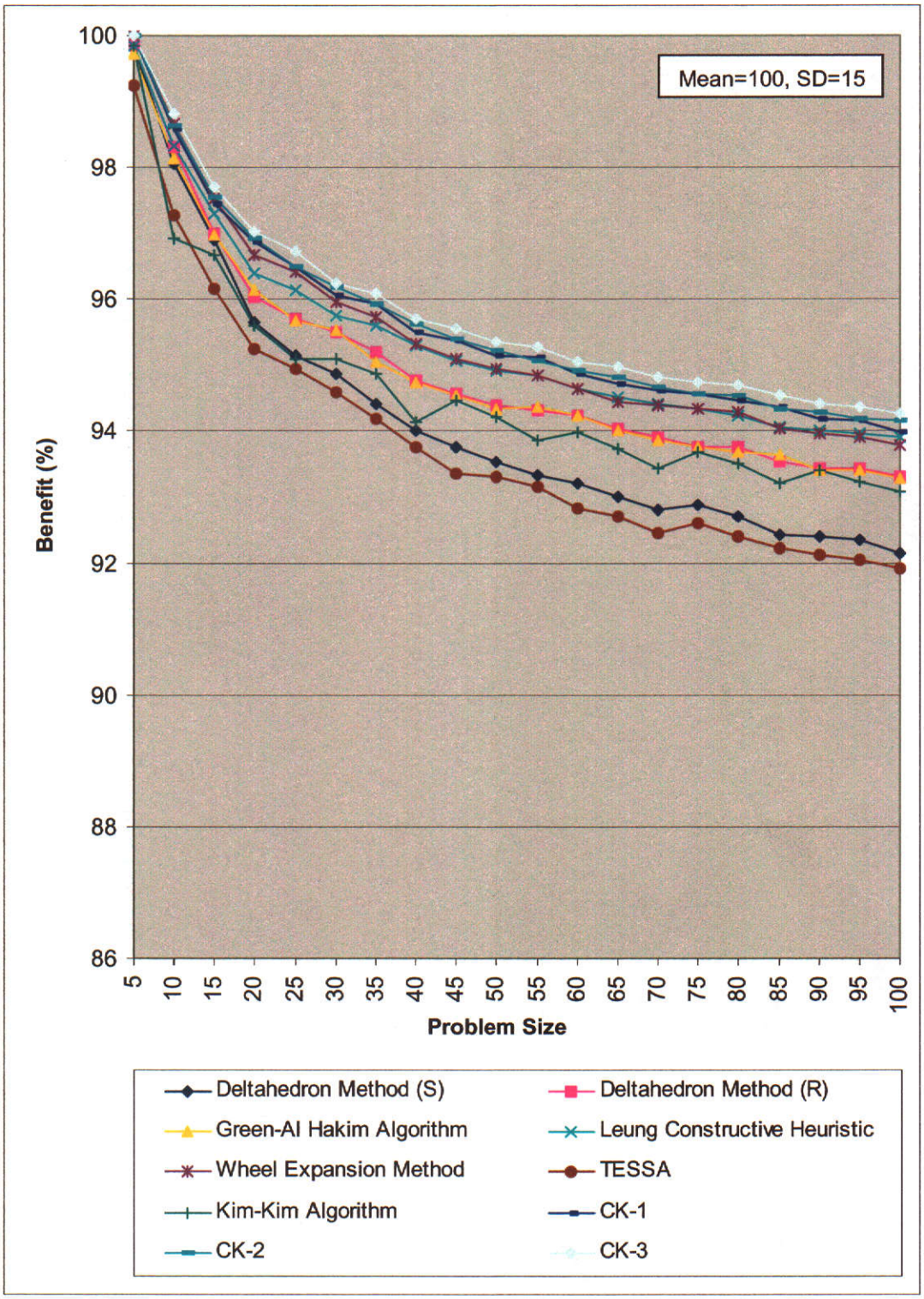


Figure 3.10. The Average Total Benefit (% of Upper Bound, Mean=100, SD=15).

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.81	99.81	99.66	99.81	100.00	98.98	99.81	100.00	100.00	100.00
10	97.42	97.78	97.60	97.91	98.28	96.70	96.13	98.17	98.19	98.51
15	96.12	96.01	96.01	96.52	96.89	95.08	95.84	96.76	96.89	97.07
20	94.44	94.88	95.03	95.38	95.80	94.07	94.36	95.92	96.08	96.16
25	93.92	94.54	94.58	95.22	95.44	93.66	93.79	95.48	95.52	95.89
30	93.42	94.32	94.25	94.67	94.85	92.96	93.95	95.04	95.10	95.28
35	92.95	93.89	93.66	94.43	94.56	92.70	93.48	94.78	94.88	95.02
40	92.41	93.45	93.32	94.06	94.13	92.30	92.76	94.40	94.52	94.59
45	92.08	93.21	93.10	93.86	93.88	91.89	93.03	94.24	94.29	94.39
50	92.01	93.07	92.98	93.60	93.71	91.65	92.77	93.92	94.04	94.19
55	91.58	92.83	92.85	93.53	93.52	91.36	92.35	93.88	93.89	94.10
60	91.54	92.85	92.86	93.26	93.37	91.14	92.56	93.64	93.79	93.86
65	91.30	92.54	92.50	93.19	93.15	90.81	92.15	93.44	93.57	93.77
70	91.06	92.39	92.40	93.06	93.01	90.80	91.83	93.25	93.29	93.59
75	91.07	92.25	92.28	92.95	92.92	90.60	92.14	93.28	93.27	93.50
80	90.92	92.23	92.17	92.91	92.81	90.66	91.99	93.10	93.22	93.45
85	90.64	92.01	92.07	92.67	92.61	90.07	91.55	92.97	93.08	93.25
90	90.61	91.79	91.84	92.58	92.47	90.21	91.81	92.83	92.96	93.18
95	90.47	91.77	91.79	92.59	92.53	90.21	91.62	92.79	92.90	93.05
100	90.24	91.74	91.76	92.51	92.27	89.79	91.39	92.54	92.73	92.93

Table 3.6. The Average Total Benefit (% of Upper Bound, Normal Distribution with Mean=100, SD=20).

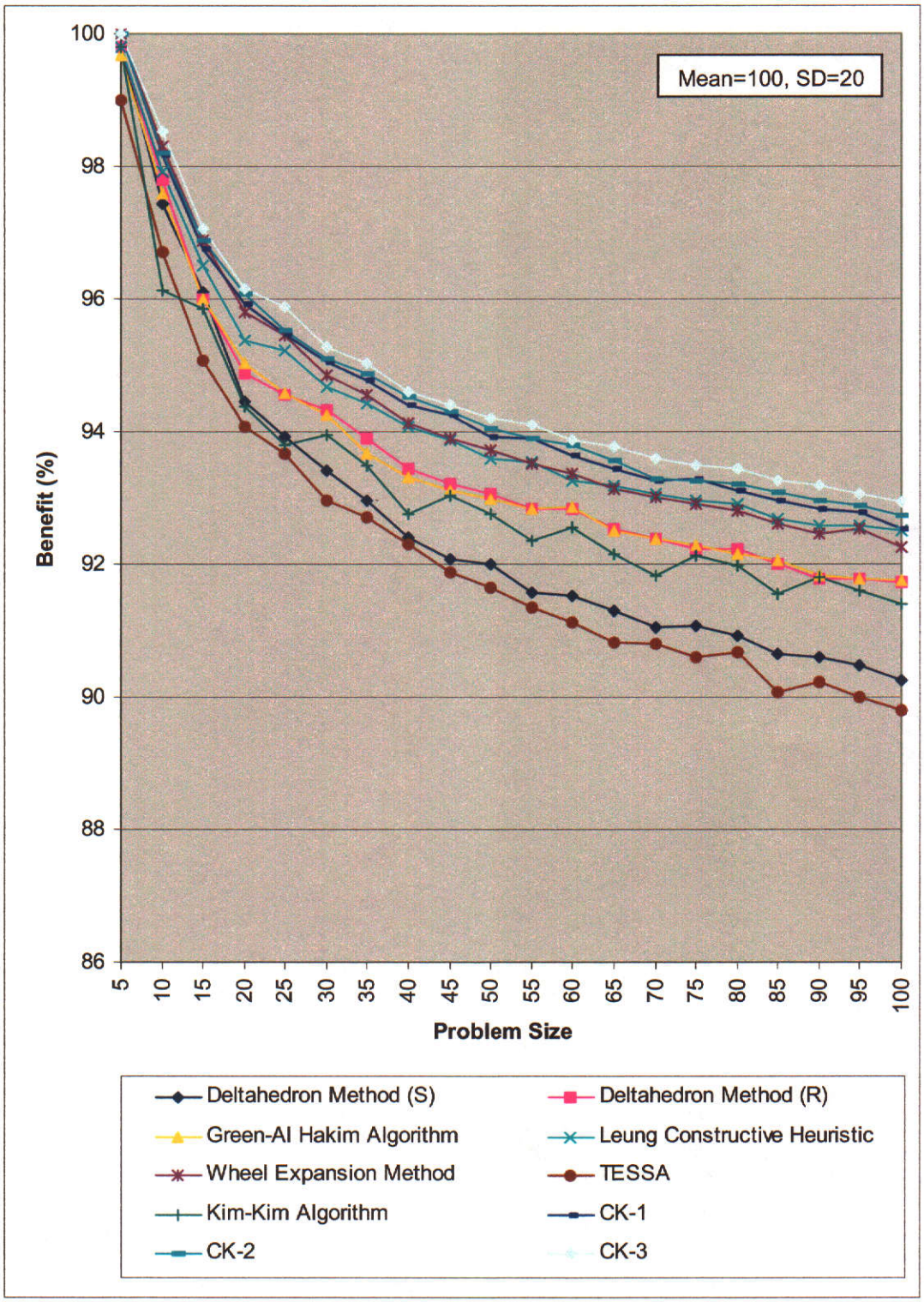


Figure 3.11. The Average Total Benefit (% of Upper Bound, Mean=100, SD=20)

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.75	99.75	99.57	99.75	100.00	98.78	99.75	100.00	100.00	100.00
10	96.93	97.39	97.07	97.39	97.88	95.77	95.14	97.79	97.85	98.15
15	95.27	95.34	95.35	95.75	96.23	94.11	94.99	96.06	96.24	96.50
20	93.39	93.88	94.06	94.63	94.96	92.84	93.34	95.19	95.35	95.41
25	92.80	93.53	93.51	94.22	94.61	92.56	92.52	94.70	94.77	95.07
30	92.25	93.31	93.24	93.60	93.97	91.85	92.87	94.10	94.23	94.45
35	91.75	92.80	92.67	93.44	93.71	91.49	92.43	93.93	93.98	94.11
40	91.04	92.26	92.20	93.02	93.10	90.90	91.53	93.47	93.49	93.69
45	90.79	92.01	91.91	92.73	92.74	82.54	91.78	93.20	93.35	93.43
50	90.56	91.80	91.77	92.44	92.52	82.28	91.46	92.87	93.00	93.22
55	90.16	91.59	91.62	92.41	92.39	81.93	90.97	92.80	92.87	93.05
60	90.05	91.59	91.53	92.14	92.17	89.47	91.25	92.50	92.65	92.79
65	89.86	91.30	91.28	91.96	91.89	89.25	90.81	92.24	92.50	92.73
70	89.51	91.06	91.09	91.82	91.77	89.28	90.51	92.11	92.20	92.45
75	89.61	90.94	90.82	91.78	91.65	89.04	90.78	92.12	92.16	92.38
80	89.33	90.82	90.83	91.71	91.68	88.79	90.55	92.01	92.09	92.40
85	89.03	90.63	90.63	91.41	91.34	88.64	90.17	91.83	91.89	92.13
90	88.99	90.51	90.49	91.33	91.17	88.42	90.43	91.69	91.75	91.96
95	88.95	90.42	90.42	91.28	91.13	88.32	90.20	91.59	91.69	91.92
100	88.63	90.25	90.22	91.23	91.03	88.14	89.95	91.44	91.55	91.78

Table 3.7. The Average Total Benefit (% of Upper Bound, Normal Distribution with Mean=100, SD=25).

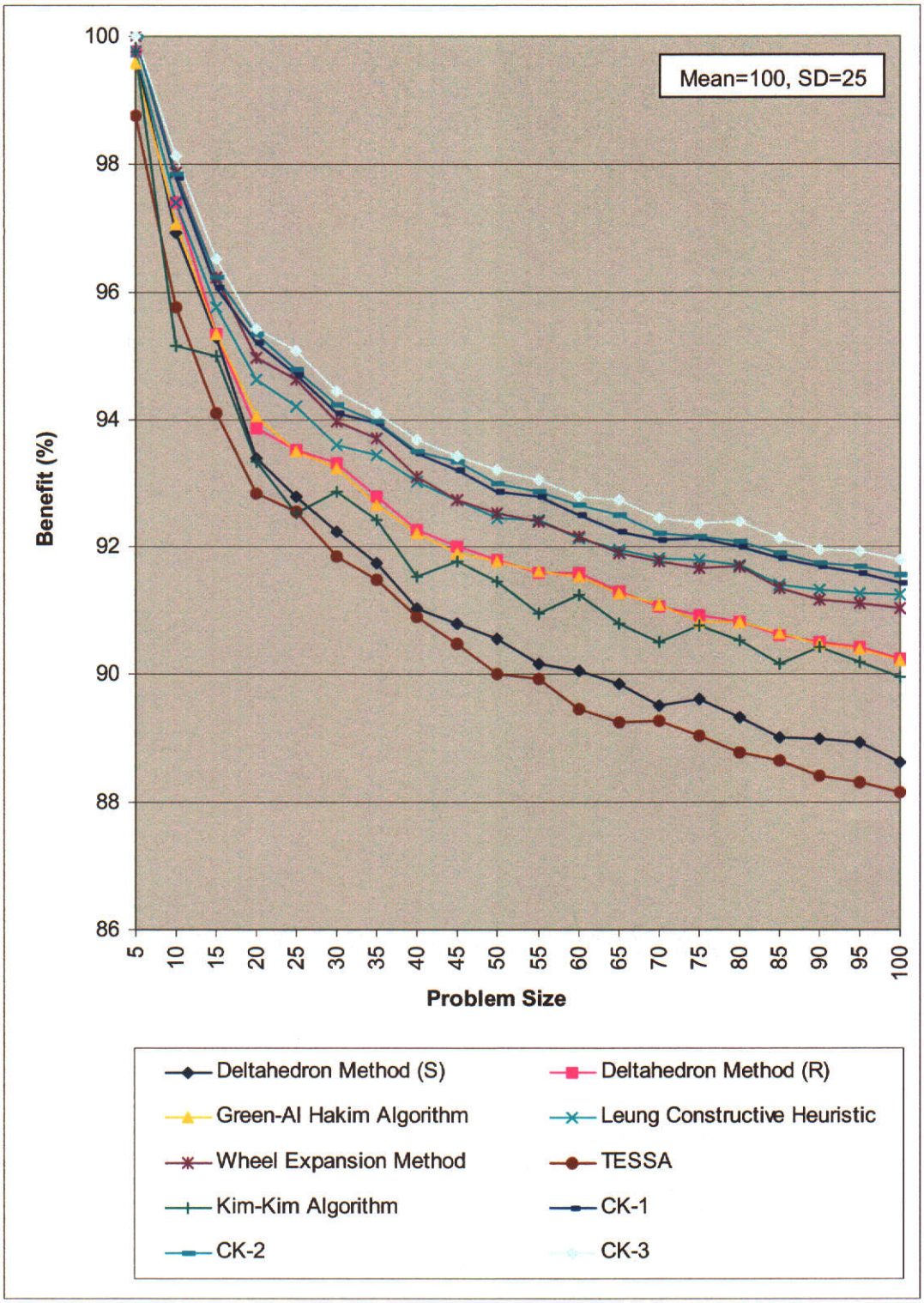


Figure 3.12. The Average Total Benefit (% of Upper Bound, Mean=100, SD=25)

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	99.70	99.70	99.47	99.70	100.00	98.55	99.70	100.00	100.00	100.00
10	96.43	96.82	96.60	96.86	97.51	95.21	94.34	97.43	97.49	97.88
15	94.51	94.60	94.63	95.09	95.57	93.23	94.32	95.47	95.64	95.96
20	92.44	93.05	93.25	93.83	94.23	91.52	92.29	94.51	94.56	94.75
25	91.87	92.69	92.73	93.37	93.82	91.50	91.48	93.89	94.01	94.36
30	91.29	92.41	92.36	92.79	93.06	90.62	91.79	93.31	93.46	93.58
35	90.68	91.76	91.68	92.54	92.75	90.43	91.31	93.13	93.16	93.31
40	89.87	91.27	91.13	92.03	92.23	89.37	90.31	92.53	92.67	92.82
45	89.57	90.93	90.79	91.75	91.76	89.25	90.73	92.19	92.44	92.57
50	89.26	90.77	90.66	91.50	91.55	88.86	90.36	91.85	92.08	92.32
55	88.92	90.46	90.48	91.42	91.39	88.74	89.75	91.89	91.89	92.19
60	88.77	90.47	90.43	91.10	91.08	88.15	90.09	91.51	91.68	91.91
65	88.45	90.17	90.06	90.89	90.92	88.15	89.58	91.30	91.47	91.83
70	88.12	90.00	89.93	90.77	90.72	87.72	89.27	91.17	91.21	91.52
75	88.29	89.73	89.74	90.72	90.64	87.45	89.53	91.10	91.13	91.38
80	87.91	89.74	89.64	90.69	90.62	87.36	89.32	90.97	91.10	91.37
85	87.76	89.45	89.51	90.38	90.28	87.04	88.95	90.76	90.84	91.11
90	87.56	89.20	89.32	90.25	90.10	86.80	89.24	90.57	90.77	90.96
95	87.55	89.23	89.23	90.22	90.52	86.60	88.94	90.49	90.61	90.86
100	87.20	89.12	89.09	90.09	89.82	86.42	88.65	90.30	90.46	90.78

Table 3.8. The Average Total Benefit (% of Upper Bound, Normal Distribution with Mean=100, SD=30).

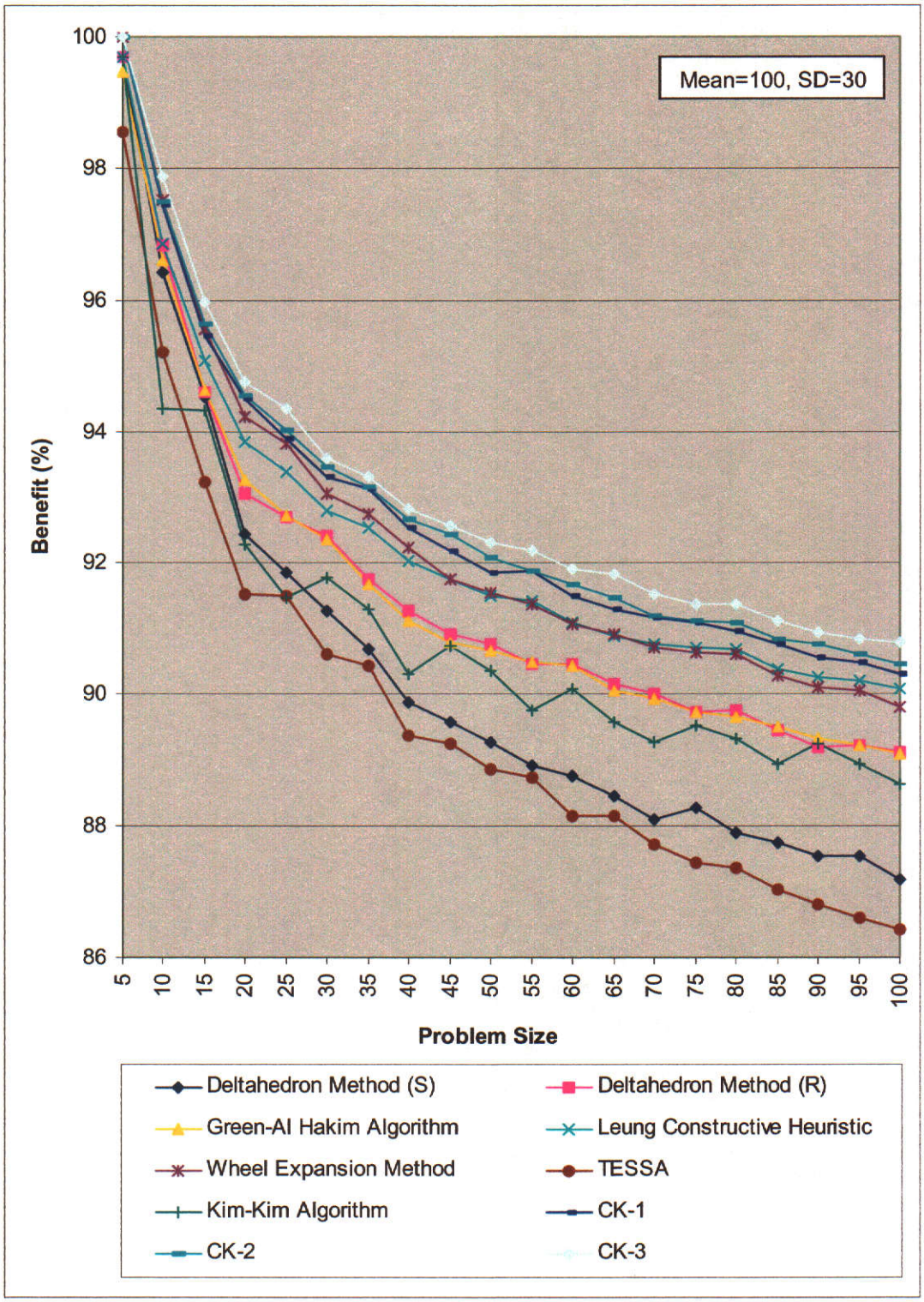


Figure 3.13. The Average Total Benefit (% of Upper Bound, Mean=100, SD=30)

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.02	0.07	0.11	0.00	0.00	0.07	0.03
20	0.02	0.02	0.02	0.18	0.14	0.92	0.09	0.07	0.17	0.08
25	0.04	0.07	0.07	0.60	1.25	2.84	0.27	0.15	0.27	0.23
30	0.06	0.17	0.16	0.85	1.89	5.00	0.41	0.33	0.57	0.45
35	0.13	0.40	0.37	1.62	2.16	10.29	0.76	0.61	0.83	0.89
40	0.27	0.89	0.88	1.82	5.08	17.38	1.02	0.86	1.50	1.65
45	0.41	1.59	1.50	3.95	8.24	33.28	1.85	1.24	2.50	2.90
50	0.70	3.00	2.91	5.87	16.26	73.27	3.37	1.88	3.93	4.68
55	0.87	4.19	4.01	9.45	21.69	105.22	5.27	3.24	4.83	6.10
60	1.45	7.19	7.07	15.17	39.28	213.86	8.38	4.92	7.33	9.23
65	1.74	9.83	9.33	22.03	50.50	302.36	12.45	6.49	9.67	12.24
70	2.72	15.67	15.51	33.56	85.85	554.05	18.99	8.93	13.43	17.22
75	3.33	21.11	20.23	48.84	121.14	779.87	26.73	10.75	19.17	24.04
80	4.01	27.71	27.33	66.03	143.43	1118.56	37.56	13.38	23.30	29.70
85	5.81	40.57	40.41	94.47	236.94	1896.31	53.40	19.45	33.43	41.47
90	7.81	59.97	59.73	127.92	314.75	2699.01	71.02	29.31	41.03	52.79
95	8.02	62.60	62.52	156.90	318.79	2952.36	87.21	35.99	45.33	57.89
100	11.99	93.81	93.65	215.03	532.42	4590.45	103.23	42.60	63.57	81.21

Table 3.9. The Average Total CPU Time (Seconds, Uniform Distribution [0,100]).

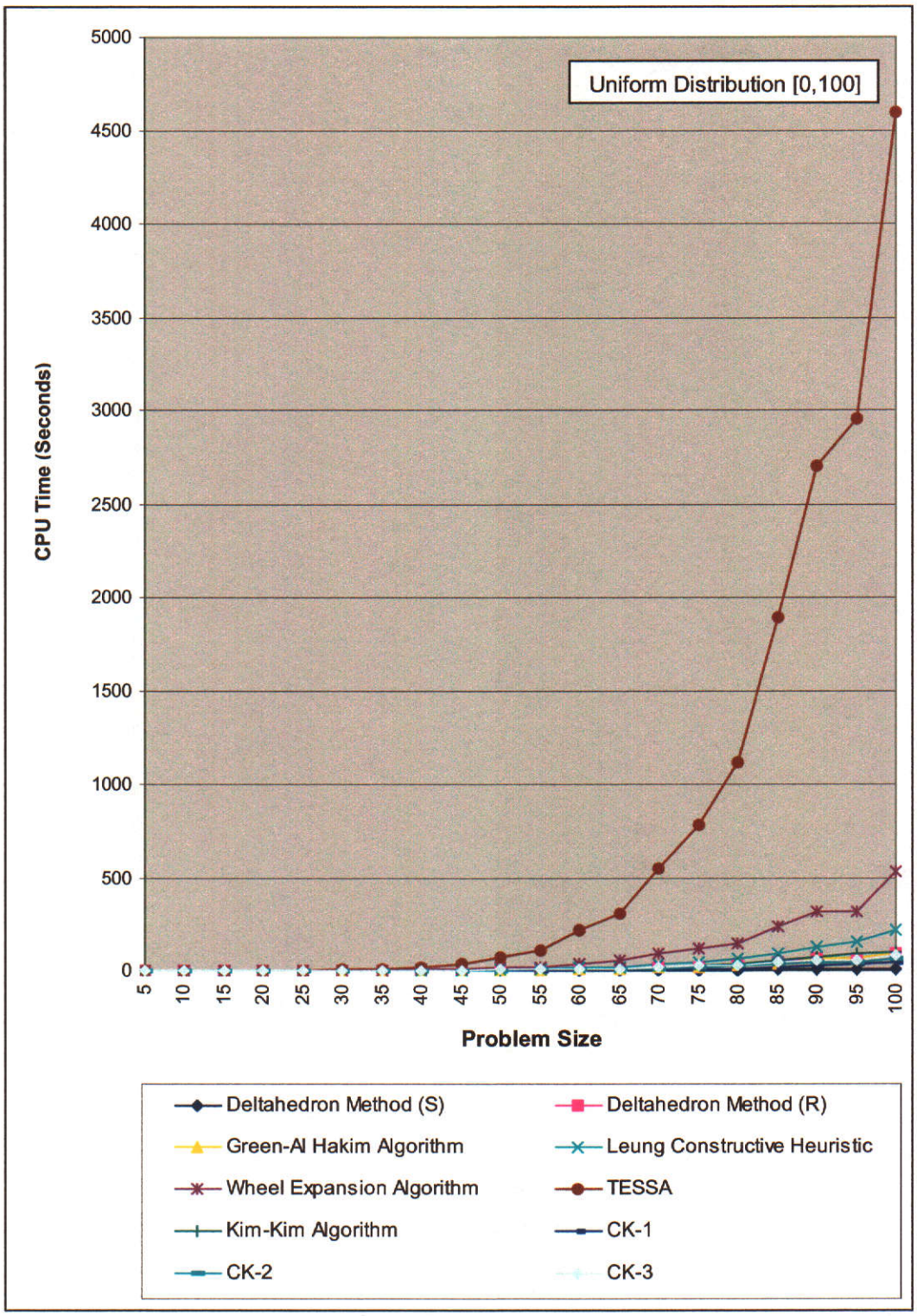


Figure 3.14. The Average Total CPU Time (Seconds).

n	Algorithms									
	Deltahedron Method (S)	Deltahedron Method (R)	Green-Al Hakim Algorithm	Leung Constructive Heuristic	Wheel Expansion Method	TESSA	Kim-Kim Algorithm	CK-1	CK-2	CK-3
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.03	0.00
15	0.00	0.01	0.00	0.01	0.06	0.08	0.01	0.03	0.07	0.02
20	0.02	0.04	0.02	0.06	0.27	0.32	0.03	0.07	0.17	0.08
25	0.04	0.08	0.08	0.20	0.83	1.24	0.11	0.13	0.27	0.24
30	0.07	0.18	0.17	0.50	1.86	3.78	0.26	0.27	0.57	0.47
35	0.14	0.42	0.40	1.16	4.08	11.75	0.60	0.40	0.83	0.93
40	0.28	0.92	0.90	2.30	8.57	39.07	1.26	0.73	1.50	1.68
45	0.43	1.61	1.52	4.50	14.11	43.70	2.30	1.17	2.50	2.91
50	0.71	3.04	2.94	7.97	25.95	95.12	4.11	1.80	3.93	4.70
55	0.90	4.22	4.03	12.64	37.59	137.34	6.56	2.47	4.83	6.11
60	1.47	7.20	7.10	20.00	56.72	236.07	10.27	3.70	7.33	9.22
65	1.77	9.85	9.35	29.47	86.45	368.87	15.32	4.83	9.67	12.26
70	2.75	15.70	15.56	43.82	163.43	576/39	23.22	6.80	13.43	17.26
75	3.35	21.15	20.25	62.98	212.69	900.56	32.80	8.97	19.17	23.96
80	4.04	27.72	27.34	86.49	292.09	1407.12	42.39	11.60	23.30	29.74
85	5.80	40.61	40.45	121.97	402.08	2103.08	57.85	15.83	33.43	42.50
90	7.84	60.00	59.80	152.61	482.31	3138.92	70.53	19.30	41.03	52.77
95	8.03	62.63	62.57	176.22	541.33	3652.54	81.43	22.27	45.33	57.79
100	12.03	93.83	93.68	264.33	822.88	5963.66	109.59	29.97	63.57	81.02

Table 3.10. The Average Total CPU Time (Seconds, Normal Distribution Mean=100, SD=5).

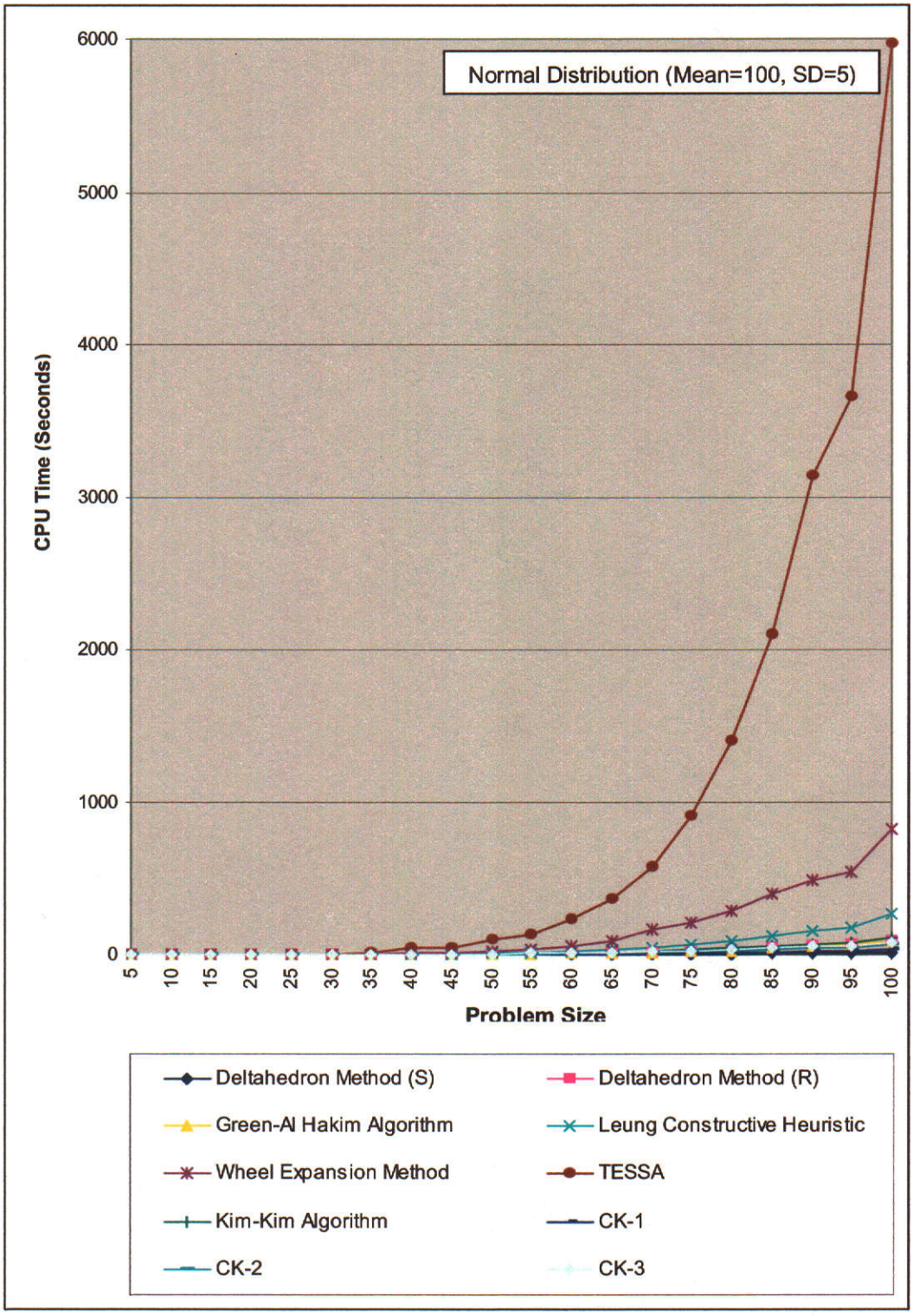


Figure 3.15. The Average Total CPU Time (Normal Distribution, Mean=100, SD=5).

3.3. Discussions and Conclusions

The construction of some graph theoretic based heuristics discussed in the earlier chapter involves successive iterations of a vertex insertion producing a maximal planar graph (MPG) with minimum degree three. There are some MPGs with minimum degree four or five, such as octahedron and icosahedron, which also exist and may represent optimal solutions. This type of MPG, however, cannot be produced by such methods. MPGs with minimum degree four or five also exist and these graphs may represent optimal solutions

In our heuristics, there are vertices with minimum degree more than 3 produced in the process of double vertex insertion. Operation O_2, O_3, and O_4 produce a vertex of degree 4, whilst Operation O_5 produces a vertex of degree 5. This means the heuristic we develop can construct any MPG, including octahedron and icosahedron.

Different from other heuristics, our heuristics have a capability to remove a poor-weighted edge from a partial solution in each insertion process. Insertion operations O_2, O_4, and O_5 allow edge removal from the current partial solution, which overcome the main drawback encountered in the established algorithms.

CHAPTER 4

GENERATING INITIAL SOLUTIONS

Some of the graph theoretic heuristics are very sensitive to the initial solution. This was first observed by Al Hakim (1991). Further, an initial solution, which is good for one problem, may not be good for others. It has been observed, that different initial solutions can yield quite different final solutions. Eades *et al.* (1982), Foulds (1983), Foulds and Giffin (1985) have underlined the need to find a best initial solution in some heuristics. This impacts on their usefulness. There are many possible subgraphs which can be used as an initial solution. If the number of vertices is n and we consider constructing a K_4 as an initial solution, then we need to consider $n(n\text{-}1)(n\text{-}2)(n\text{-}3)/24$ possible sets of 4 vertices. All four vertices are mutually adjacent, and so all these possibilities are available.

In this Chapter we discuss several initial solutions used in the established graph theoretic based heuristics, most of which are the best K_3, a K_4, or the best K_4. We also investigate the sensitivity of some graph theoretic based heuristics to different initial solutions, to find out the quality of the final solution and the sensitivity of the algorithms to different initial solutions. In Section 4.1 several different initial solutions are proposed. In Section 4.2 we present several computational experiments involving the various initial solutions. We compare the performance of the existing algorithms when each different type of initial solution is used. In Section 4.3 we provide the computational work for a number of heuristics followed by their performance presented in tables and charts.

4.1 The Established Initial Solutions

Most of the initial solutions used in the existing graph based heuristics are K_3 or K_4. This means that the initial solution considers all $\binom{n}{3}$ or $\binom{n}{4}$ candidates and chooses the best (highest weight). Since the amount of computation to obtain a K_3 in this process is proportional to $n(n\text{-}1)(n\text{-}2)$, the complexity is bounded by $O(n^3)$. Similarly for a K_4 the complexity is bounded by $O(n^4)$. From this solution, a partial solution is then grown iteratively by certain insertion operations.

The following table summarizes the established initial solutions used in the main graph theoretic based heuristics.

Algorithms	Initial Solutions
1. Deltahedron Method (S-Construction)	K_4
2. Deltahedron Method (R-Construction)	Best K_4
3. Green-Al Hakim	Best K_3
4. Leung's Constructive Heuristic	Best K_4
5. Wheel Expansion Algorithm	K_4
6. Kim-Kim Algorithm	Best K_4
7. TESSA	Best K_3
8. Algorithm CK-1	Best K_4
9. Algorithm CK-2	Best K_4
10. Algorithm CK-3	Best K_4

Table 4.1

Some of the graph theoretic based heuristics are very sensitive to the initial solution. It might happen that the second or even the third best initial solution, can produce a final solution, which has a higher weight than the final solution produced by using the best initial solutions. To see this, consider the set of data with n=10, displayed in Figure 4.1.

0	44	60	5	68	61	77	22	86	78
44	0	0	40	8	1	17	57	26	18
60	0	0	34	74	43	35	51	91	60
5	40	34	0	53	69	14	77	70	86
68	8	74	53	0	31	0	87	9	48
61	1	43	69	31	0	17	10	26	66
77	17	35	14	0	17	0	34	27	43
22	57	51	77	87	10	34	0	83	52
86	26	91	70	9	26	27	83	0	44
78	18	60	86	48	66	43	52	44	0

Figure 4.1

The total number of all possible K_4's which can be selected as an initial solution is $n(n\text{-}1)(n\text{-}2)(n\text{-}3)/24$. Since we have n=10 in the above edge weights, there are 210 possible combinations of 4 vertices. We put them in the set of non-increasing order of edge weights. The first two best initial solutions are (1,3,9,10) and (4,8,9,10), with the weight 419 and 412, respectively. Suppose we use them to implement the Deltahedron Method, the Kim-Kim Algorithm, the Wheel Expansion Algorithm, the Leung's Constructive Heuristic, and our three new heuristics. Then we have the results depicted in Table 4.2.

Initial Solution		Weights of the Final Solution						
Vertices	Weight	Deltahedron Method (R-Construction)	Kim-Kim Algorithm	Wheel Expansion	Leung's Constructive Heuristic	CK-1	CK-2	CK-3
1,3,9,10	419	1443	1305	1505	1443	1505	1505	1505
4,8,9,10	412	1487	1399	1505	1486	1505	1505	1505

Table 4.2

From this table, we can see that compared to the first best initial solution, the second best initial solution in the Deltahedron Method, the Wheel Expansion Method and the Leung's Constructive Heuristic gives a higher-weighted final solution. Note that the Deltahedron Method (S-Construction) and the Wheel Expansion Method in Table 4.2 use the best K_4, instead of a K_4 defined in their initial solution formulation.

From the set of edge weights in Figure 4.1 we have $n(n\text{-}1)(n\text{-}2)/6$ possible K_3's. Since we have n=10, there are 120 possible K_3's we can select as an initial

solution. The first two best initial solutions are (1,3,9) and (4,8,9) with the weight 237 and 230. Table 4.3 displays the final solution produced by TESSA and the Green–Al Hakim algorithm using these two selections.

Initial Solution		Weights of the Final Solution	
Vertices	Weight	TESSA	The Green-Al Hakim Algorithm
1,3,9	237	1318	1443
4,8,9	230	1318	1487

Table 4.3

This table indicates that in TESSA both initial solutions give the same weight of final solution, whilst in the Green-Al Hakim algorithm the second best initial solution produces the higher-weighted final solution.

It is very interesting to see that the second best initial solution can give a higher-weighted final solution than the best initial solution does. Further, two initial solutions with the same weight can produce different-weighted final solutions. To see this, consider the set of data with n=25, displayed in Figure 4.2.

```
 0 47 57 28 22 21 46 57 56  6 41  8 28  4 12 54 52  1  4 15 55 56 34 24 29
47  0 52 37 22 60  3 44 38 15 58 57 38 58 49  4 33  3 48 39 19 31 16 52 55
57 52  0 30 52  8 38 40 41 12 29  8 46 13  4 59 46  3 30 30 27 26 22 50 29
28 37 30  0 16 19 38 51 15 40 48 46 53 55  2  8 49 14  8 31  5 16 26 11 14
22 22 52 16  0  0 58 42 30  9 17 30 45 35 14 54 23 18 45 53 23 46 28 12 47
21 60  8 19  0  0 17 37 18 59 61 11 17  1  0 14 51 22 14 11 32  9 50 47 25
46  3 38 38 58 17  0 11 43 45 17  5  1 18 23 43 27  8 13 58 24 21 17 12 36
57 44 40 51 42 37 11  0 43  9 60 21 53 30 20 53 59  3 12 21 35 21 28 48  3
56 38 41 15 30 18 43 43  0  1 47  7  2 51 40 23  3 37  4 51 11 39 44 13 33
 6 15 12 40  9 59 45  9  1  0 42  5 27  1 40 33 39  4 23 28 19 20 42 23 36
41 58 29 48 17 61 17 60 47 42  0 54 45 27 40 30 38  0 50  8  6 23  0 36  0
 8 57  8 46 30 11  5 21  7  5 54  0 18 52 26 47 32 45  6 31 34 60 47 35 46
28 38 46 53 45 17  1 53  2 27 45 18  0 10  3 55 12 43 16 34 46 39 47 14 36
 4 58 13 55 35  1 18 30 51  1 27 52 10  0 14 26 40 59 21 20 13 40 15 30 12
12 49  4  2 14  0 23 20 40 40 40 26  3 14  0 18 34 17 55 27 51  4  5 19 27
54  4 59  8 54 14 43 53 23 33 30 47 55 26 18  0 15 48 21 10 39 11 57 53  5
52 33 46 49 23 51 27 59  3 39 38 32 12 40 34 15  0 37 36  7 52 15 12  3 26
 1  3  3 14 18 22  8  3 37  4  0 45 43 59 17 48 37  0 29 23 54  1 36 32 35
 4 48 30  8 45 14 13 12  4 23 50  6 16 21 55 21 36 29  0 32  2 15 22 39 55
15 39 30 31 53 11 58 21 51 28  8 31 34 20 27 10  7 23 32  0 48  0  0 43 34
55 19 27  5 23 32 24 35 11 19  6 34 46 13 51 39 52 54  2 48  0 42 38 43 42
56 31 26 16 46  9 21 21 39 20 23 60 39 40  4 11 15  1 15  0 42  0 16  1 29
34 16 22 26 28 50 17 28 44 42  0 47 47 15  5 57 12 36 22  0 38 16  0 59 10
24 52 50 11 12 47 12 48 13 23 36 35 14 30 19 53  3 32 39 43 43  1 59  0 46
29 55 29 14 47 25 36  3 33 36  0 46 36 12 27  5 26 35 55 34 42 29 10 46  0
```

Figure 4.2

From the above data we have two best K_4's which have the same weight 320, namely (1,3,8,16) and (2,6,8,11). Suppose we employ each of these to the graph theoretic based heuristics which use K_4 or the best K_4 as their initial solution. Then we have the following table, which depicts the weight of the final solutions produced by each algorithm.

Initial Solution (Weight = 320)	Weights of the Final Solution							
	Deltahedron Method (S)	Deltahedron Method (R)	Kim-Kim Algorithm	Wheel Expansion	Leung Constructive Heuristic	CK-1	CK-2	CK-3
(1,3,8,16)	2850	3281	3179	3256	3305	3306	3274	3299
(2,6,8,11)	3025	3296	3108	3270	3276	3260	3293	3338

Table 4.4

Since TESSA and the Green-Al Hakim algorithm use the best K_3 as an initial solution, there may be several different constructions of K_3 of the same weight, as ties are broken arbitrarily is taken in vertex-selection decision. These different selections often produce different final solutions with quite different weight, if the algorithm is very sensitive to the initial solution. The following example shows how two solutions given by TESSA, using two different initial solutions, are very different. Consider the following data set.

0	79	7	21	2	89	43	69	36	84	20	18	16	77	35
79	0	1	42	74	2	20	59	82	74	71	4	46	44	87
7	1	0	87	69	19	34	35	19	0	58	1	36	78	23
21	42	87	0	40	36	55	8	27	91	68	23	92	63	58
2	74	69	40	0	64	24	87	6	30	25	53	29	12	60
89	2	19	36	64	0	72	52	16	19	11	20	7	55	71
43	20	34	55	24	72	0	16	8	50	14	80	61	79	48
69	59	35	8	87	52	16	0	86	91	34	46	73	4	77
36	82	19	27	6	16	8	86	0	86	28	45	37	91	16
84	74	0	91	30	19	50	91	86	0	89	88	7	61	67
20	71	58	68	25	11	14	34	28	89	0	86	80	72	40
18	4	1	23	53	20	80	46	45	88	86	0	54	13	85
16	46	36	92	29	7	61	73	37	7	80	54	0	26	14
77	44	78	63	12	55	79	6	91	61	72	13	26	0	87
77	44	78	63	12	55	79	6	91	61	72	13	26	0	87

Figure 4.3

Since n=15, there are 455 possible triangular faces in the set of non-increasing order of weight. In this set, the first two of triple vertices are (8,9,10) and (10,11,12), which have the same weight 263. When we meet this situation, we take one of them, as suggested in TESSA that ties are broken arbitrarily. If we take the best-weighted triangular face (10,11,12) having weight 263, we will end up with the final weighted graph of weight 2867. However, when we employ the initial solution (8,9,10), which has the same weight 263, the resulting final graph is of weight 2505 (See Table 4.5). This shows that the weight of the solution might be quite different from one to another, although their initial solution has the same weight. Application of the Green-Al Hakim algorithm to these initial solutions, respectively, yields final solution values of 2767 and 2757.

Initial Solution (weight=263)	Weights of the Final Solution	
	TESSA	Green-Al Hakim Algorithm
(8,9,10)	2505	2767
(10,11,12)	2867	2757

Table 4.5

As mentioned by Al Hakim (1994), TESSA is highly sensitive to the structure of the REL chart. The above example shows how two solutions given by TESSA, using two different initial solutions, are very different.

4.2. Generating Initial Solutions

We conducted several computational experiments involving various initial solutions and compared the performance of existing algorithms when each different type of initial solution is used. Our computational work indicated that the initial solution could influence the final solution obtained by heuristics. To test the quality and the sensitivity of the final solution to different initial solutions we propose 10 alternatives. In all cases we build up a K_4.

Initial Solution 1

We select 4 vertices randomly regardless of their sum of weight.

Initial Solution 2

We start by selecting a path on 3 vertices having the highest weight. The edge incident to one of the vertices having maximum weight is added to obtain the fourth vertex. Figure 4.4 illustrates the construction.

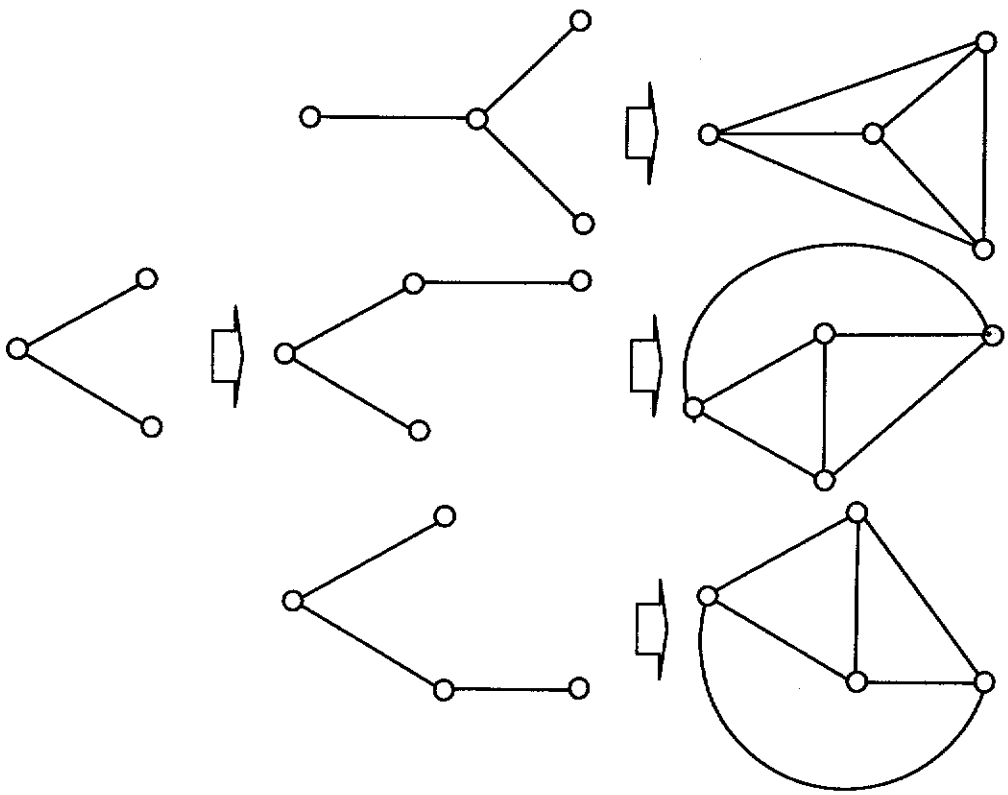


Figure 4.4

Initial Solution 3

Here three different edges having a common vertex v and the highest weight are selected. Adding appropriate edges yields the initial K_4 (see Figure 4.5).

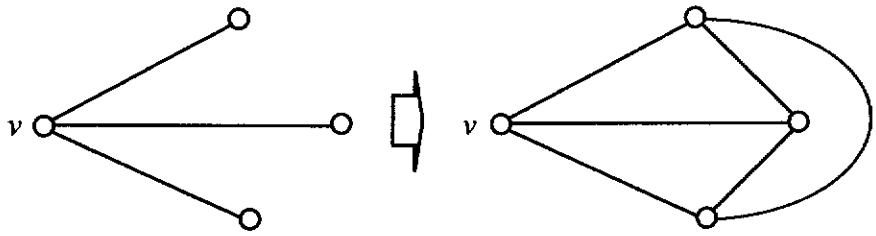


Figure 4.5

Initial Solution 4

This variation selects the pair of independent edges having the highest total weight (see Figure 4.6).

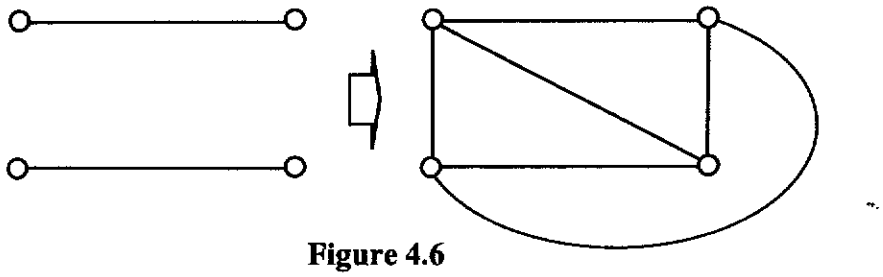


Figure 4.6

Initial Solution 5

Here we start with the highest-weighted triangle (best K_3) and apply the 1-vertex insertion operation.

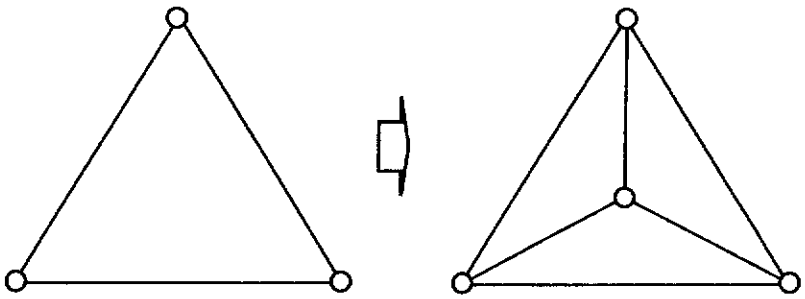


Figure 4.7

Initial Solution 6

The process of constructing an initial solution in this variation is the same as that in The Deltahedron Method (S-construction), which has been discussed in Chapter 2, Section 2.4. Here we again outline its process. In constructing an initial solution, a

list of non-increasing order of weight is created. This is done by summing up the weight of edges incidents to each vertex. The total weight of edges (i,j) incident to vertex i, can be expressed as $M(i)$, where

$$M(i)=\sum_{j} w(i,j).$$

The first four vertices are selected for an initial subgraph K_4.

Initial Solution 7

Two highest-weighted edges having a common vertex are selected. Join an edge between the two non-adjacent vertices, so a triangle is constructed. A vertex, which has the 3 highest-weighted edges incident to the existing vertices, is added.

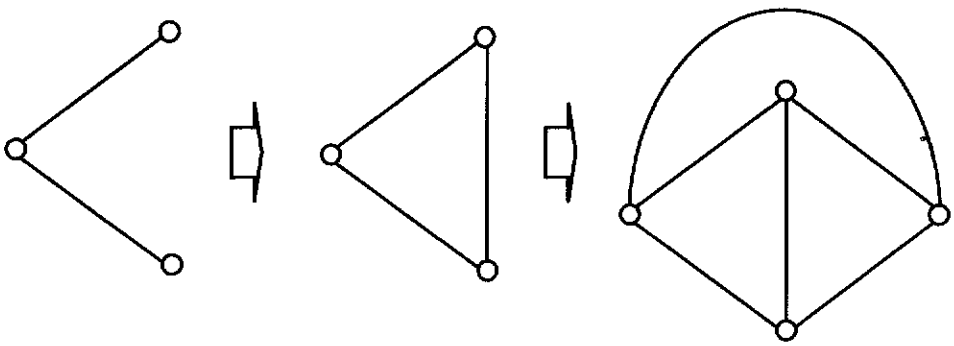


Figure 4.8

Initial Solution 8

Two vertices, which have the highest-weighted edge, are selected. Two other vertices, which have the 5 highest-weighted edges incident to the existing vertices, are also selected to construct a K_4.

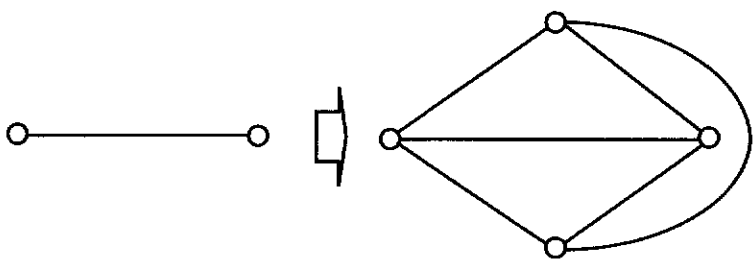


Figure 4.9

Initial Solution 9

Two vertices, which have the highest-weighted edge, are selected. Then, a vertex, having 2 highest-weighted edges to the existing vertices, is added; so a triangle is constructed. The fourth vertex, which has 3 highest weighted edge, is then inserted to create a K_4 (see Figure 4.10).

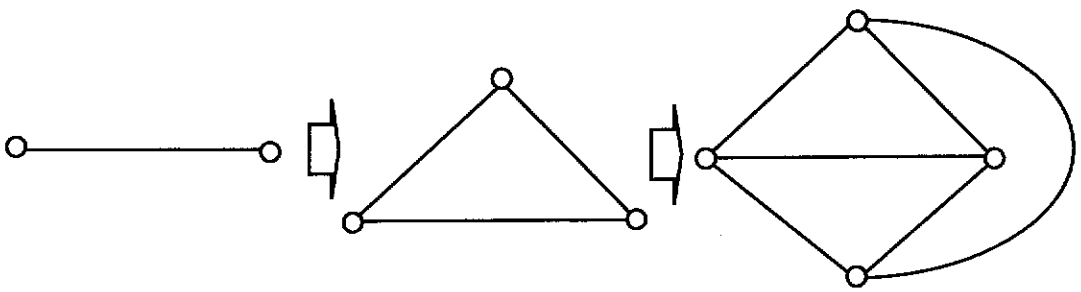


Figure 4.10

Initial Solution 10

Best K_4.

4.3 Computational Results

For a number of heuristics we have investigated the best initial starting solution. Ten different initial solutions have been implemented for the Deltahedron Method (R-construction), the Kim-Kim Algorithm, the Leung's Constructive Heuristic, the Wheel Expansion Method, Algorithm CK-1, Algorithm CK-2, and Algorithm CK-3. A comparative analysis of the performance of these heuristics under each starting initial solution, is carried out based on 600 randomly generated problems. We now discuss the details of the test-problem size, generation of random test problems, and implementation. The size of the test problems is represented by the number n of vertices (facilities), where n ranged from 5 to 100 in increments of 5. For each n, 30 test problems were generated by selecting the edge weights from a uniform distribution [0,100]. This results in 600 problems. The computational work of initial starting solution experiments was also carried out on a Silicon Graphic Workstation (R5000) running at clock speed of 150 MHz. All algorithms are implemented in C language.

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	97.21	99.30	99.37	98.71	99.89	99.89	99.43	99.59	99.56	99.89
10	88.16	90.83	90.89	91.15	92.28	93.13	92.31	93.16	92.78	92.62
15	87.54	89.23	88.80	88.63	90.74	90.43	90.16	90.77	90.41	90.67
20	86.60	88.90	88.27	88.50	89.89	89.74	89.15	89.51	89.89	89.88
25	86.91	88.70	88.54	88.27	89.84	89.08	89.22	89.46	89.59	89.77
30	87.57	88.78	88.53	88.14	89.77	89.02	89.48	89.65	89.70	89.72
35	87.51	88.65	88.99	88.59	89.55	89.24	89.16	89.48	89.47	89.63
40	87.83	89.00	88.81	88.78	89.51	89.17	89.40	89.74	89.63	89.60
45	88.27	89.07	89.00	88.70	90.08	89.22	89.50	89.83	89.82	90.01
50	88.70	89.24	89.32	89.24	90.02	89.49	89.90	90.05	89.79	90.05
55	88.73	89.48	89.63	89.39	89.93	89.58	90.00	90.17	90.17	90.12
60	89.16	89.74	89.85	89.77	90.36	89.79	90.27	90.35	90.26	90.49
65	89.28	90.06	90.04	89.75	90.37	89.89	90.25	90.38	90.62	90.28
70	89.36	90.03	90.20	90.05	90.58	89.94	90.42	90.42	90.48	90.56
75	89.61	90.18	90.26	89.93	90.77	90.15	90.30	90.50	90.53	90.60
80	89.79	90.37	90.39	90.15	90.80	90.17	90.57	90.71	90.63	90.78
85	90.21	90.53	90.65	90.44	91.02	90.34	90.77	90.86	90.88	90.84
90	90.07	90.62	90.50	90.43	90.93	90.49	90.81	90.91	90.83	91.03
95	90.14	90.82	90.95	90.69	91.08	90.72	91.06	91.17	91.20	91.15
100	90.57	90.89	90.95	90.59	91.25	90.55	91.09	91.29	91.20	91.24

Table 4.6. The Average Total Benefit (Deltahedron Method) (% of Upper Bound)

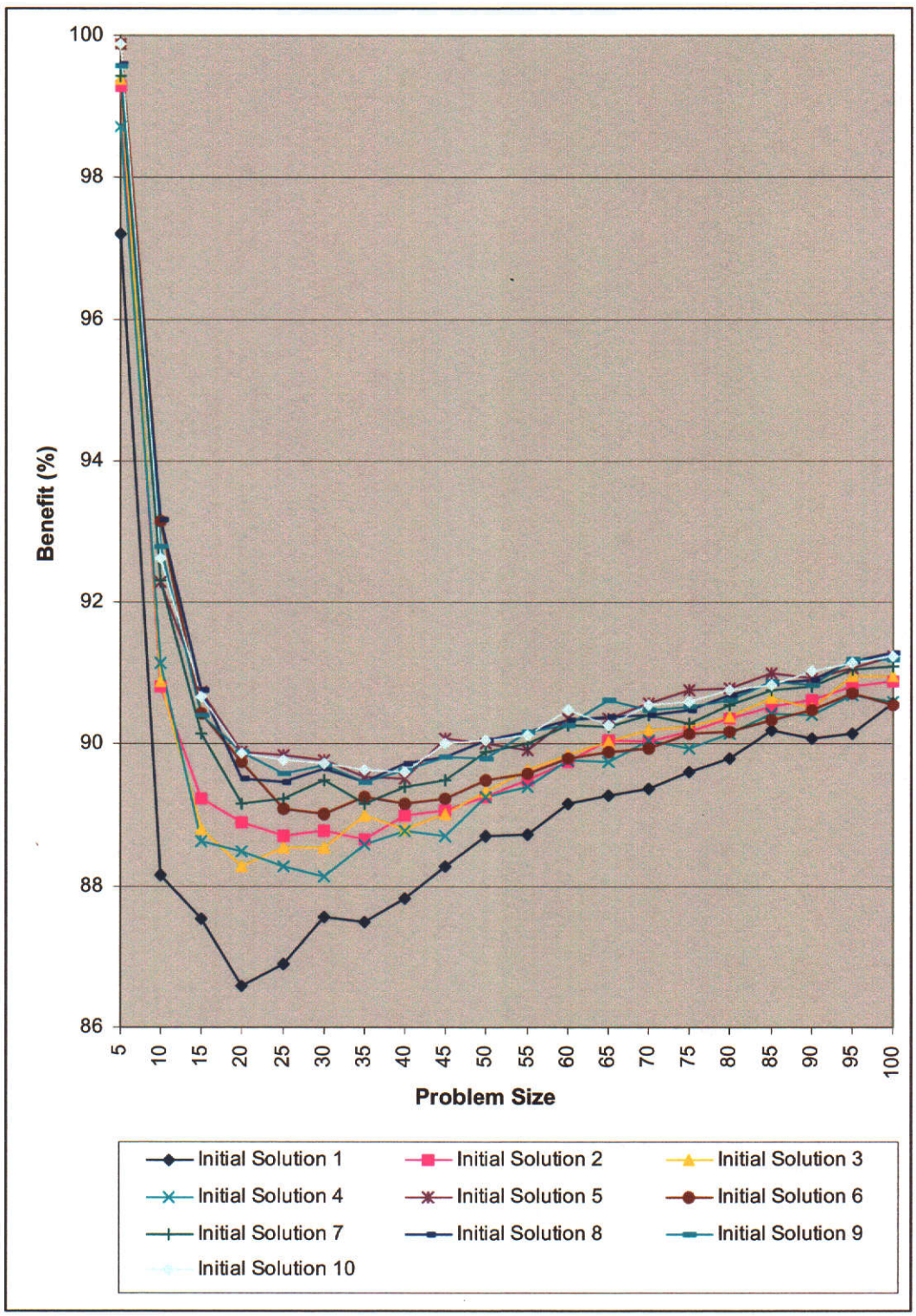


Figure 4.11. The Average Total Benefit (Deltahedron Method)

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	97.21	99.30	99.37	98.71	99.86	99.89	99.43	99.59	99.56	99.89
10	84.39	86.88	87.02	86.10	89.72	86.72	88.62	88.62	89.11	89.53
15	86.49	88.54	88.48	87.43	90.05	88.39	89.40	91.36	89.92	90.21
20	84.34	87.29	87.54	86.17	88.24	86.82	87.80	88.39	88.11	88.28
25	84.11	86.16	86.34	85.66	87.02	86.11	86.93	25.85	87.22	87.35
30	86.00	87.38	87.39	87.13	88.08	87.63	88.02	88.16	88.49	88.60
35	86.00	87.56	87.71	87.13	88.24	87.14	88.20	88.45	88.30	88.46
40	85.80	86.89	87.02	86.79	87.71	87.00	87.50	87.77	87.66	88.15
45	87.39	88.18	88.21	87.96	88.77	87.95	88.59	88.90	88.92	88.84
50	87.14	88.02	88.10	88.03	88.75	87.68	88.50	88.66	88.69	88.87
55	87.09	87.70	87.83	87.65	88.38	87.83	88.13	88.44	88.20	88.78
60	88.11	88.86	88.79	88.65	89.11	88.62	89.00	89.26	89.25	89.34
65	87.99	88.68	88.71	88.65	89.28	88.57	89.14	89.18	89.22	89.22
70	88.00	88.62	88.73	88.41	89.10	88.52	88.66	89.14	89.04	89.19
75	88.85	89.36	89.31	89.02	89.89	89.15	89.73	89.76	89.84	89.99
80	88.68	89.34	89.22	89.11	89.64	89.04	89.63	89.62	89.60	89.79
85	88.68	89.09	89.36	88.87	89.59	88.89	89.47	89.46	89.55	89.64
90	89.20	89.79	89.66	89.61	90.21	89.51	90.01	90.21	90.21	90.13
95	89.26	89.93	89.85	89.65	90.18	89.68	90.12	90.24	90.21	90.15
100	89.24	89.55	89.69	89.71	90.06	89.36	89.89	90.01	90.08	90.10

Table 4.7. The Average Total Benefit (Kim and Kim Algorithm) (% of Upper Bound)

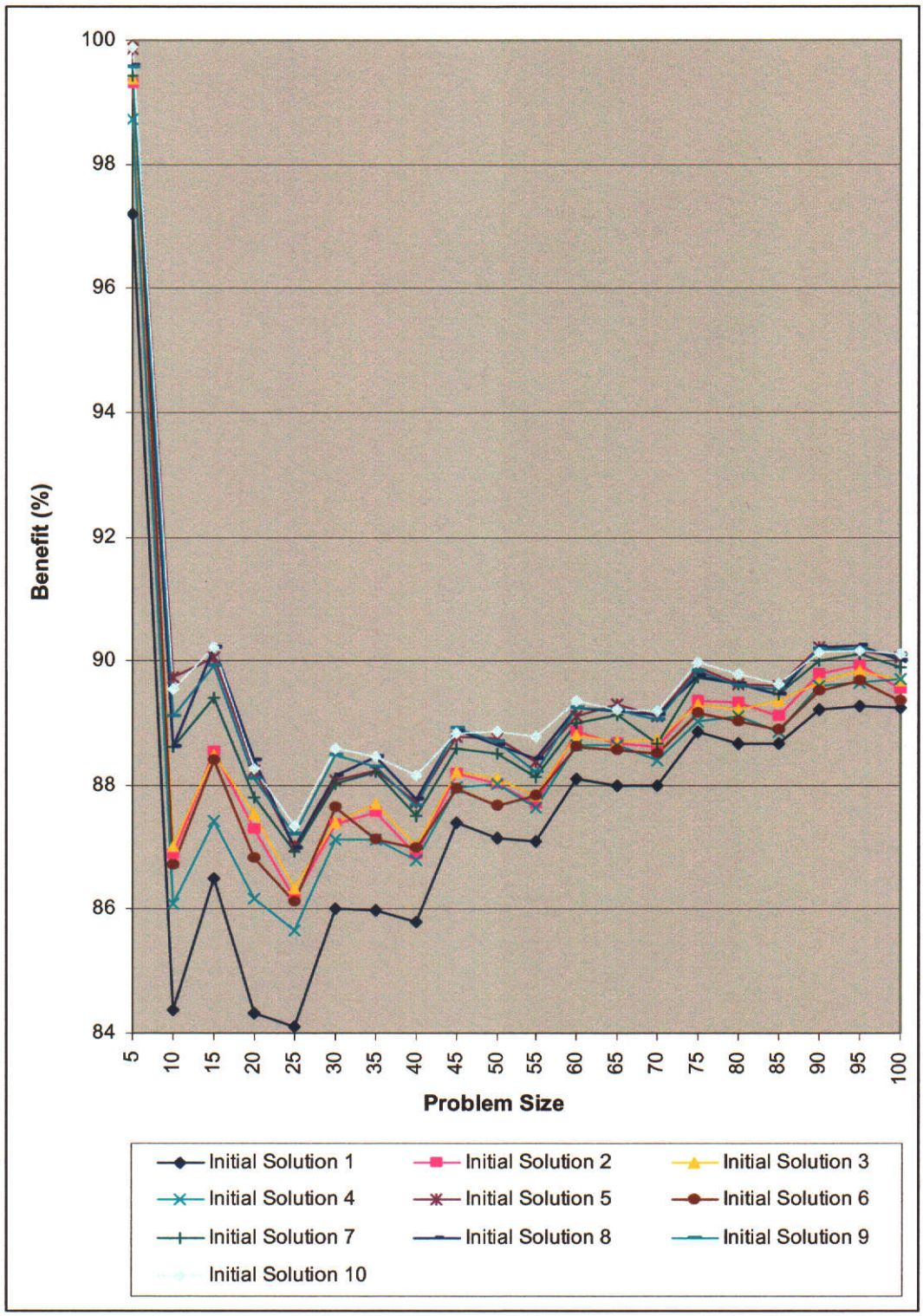


Figure 4.12. The Average Total Benefit (Kim-Kim Algorithm)

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	97.21	99.30	99.37	98.71	99.86	99.89	99.43	99.59	99.56	99.89
10	88.62	91.85	91.84	91.31	93.16	93.53	92.84	93.59	93.26	93.72
15	88.12	90.01	89.64	89.50	91.70	90.96	90.86	91.36	91.46	91.63
20	87.32	89.71	89.40	88.92	90.86	89.84	90.39	90.67	90.62	90.98
25	87.57	89.49	89.59	88.94	90.58	89.79	90.10	90.41	90.45	90.65
30	88.08	89.58	89.46	89.23	90.54	89.82	90.19	90.51	90.42	90.66
35	88.50	90.00	90.06	89.55	90.45	89.83	90.17	90.50	90.43	90.72
40	88.84	89.74	89.90	89.60	90.70	89.88	90.22	90.46	90.42	90.71
45	89.29	89.98	89.95	89.62	90.80	89.99	90.27	90.73	90.67	90.77
50	89.29	90.06	90.28	90.05	90.89	90.14	90.70	80.28	90.54	90.69
55	89.57	90.55	90.66	90.39	91.04	90.38	90.83	90.97	90.93	90.98
60	89.90	90.56	90.58	90.26	91.11	90.36	90.93	91.01	91.01	91.24
65	90.13	90.74	90.75	90.80	91.25	90.70	91.06	91.24	91.26	91.32
70	90.25	90.98	90.97	90.77	91.30	90.72	91.27	91.50	91.44	91.50
75	90.48	91.01	90.92	90.95	91.50	91.06	91.38	91.43	91.37	91.57
80	90.57	91.10	91.11	90.99	91.50	90.77	91.49	91.64	91.57	91.58
85	90.95	91.40	91.49	91.07	91.71	91.25	91.65	91.77	91.77	91.79
90	90.87	91.48	91.47	91.20	91.84	91.24	91.62	91.73	91.76	89.09
95	91.05	91.61	91.58	91.55	92.01	91.47	91.95	91.96	91.90	92.04
100	91.39	91.76	91.67	91.47	92.07	91.45	91.92	92.03	92.03	92.03

Table 4.8. The Average Total Benefit (Leung's Constructive Heuristic)(% of Upper Bound)

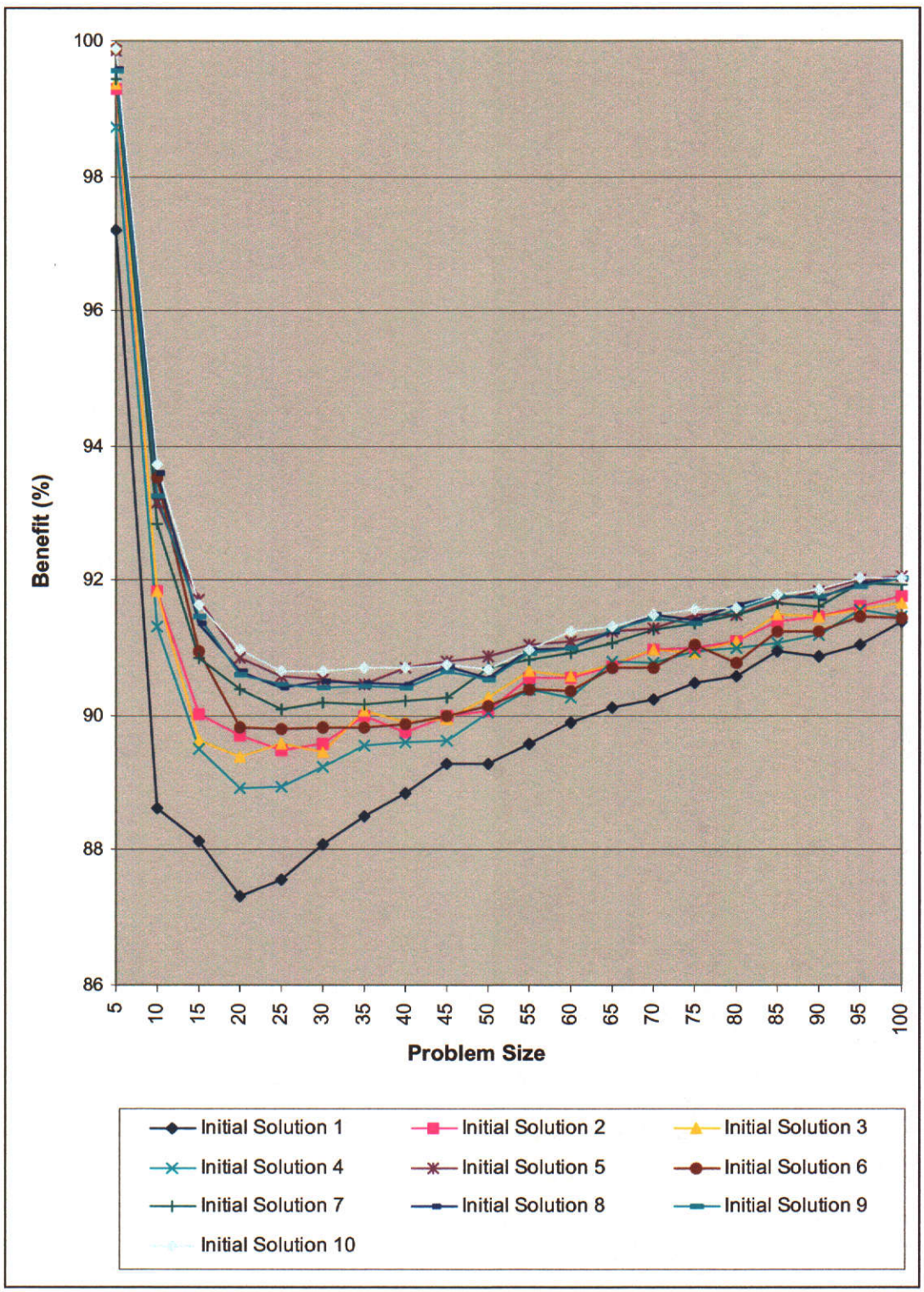


Figure 4.13. The Average Total Benefit (Leung's Constructive Heuristic)

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	93.62	94.34	94.12	94.44	94.25	94.40	94.31	94.58	94.29	94.19
15	91.48	91.40	91.43	92.28	91.83	91.82	91.89	91.96	91.85	91.64
20	90.47	90.83	90.48	90.72	90.26	90.26	90.59	90.59	90.69	90.52
25	90.06	90.13	90.12	90.73	90.53	89.84	90.17	90.25	90.28	90.24
30	89.98	90.42	90.43	90.49	90.60	90.05	90.52	90.39	90.43	90.51
35	89.74	90.29	90.28	90.07	90.15	90.05	90.13	89.89	90.13	90.43
40	90.14	90.25	90.14	90.24	90.08	90.07	90.13	90.36	90.28	90.16
45	90.06	90.38	90.46	90.48	90.59	89.86	90.40	90.51	90.51	90.53
50	90.41	90.38	90.38	90.65	90.44	90.17	90.43	90.51	90.35	90.48
55	90.39	90.62	90.59	90.41	90.47	90.49	90.45	90.40	90.51	90.63
60	90.49	90.55	90.56	90.59	90.71	90.48	90.64	90.74	90.69	90.82
65	90.71	90.75	90.82	90.86	90.89	90.51	90.72	90.84	90.76	90.86
70	90.65	90.95	91.01	90.80	90.98	90.76	90.99	91.00	90.83	91.06
75	90.83	90.89	91.00	90.83	91.03	90.72	90.94	91.03	91.06	91.11
80	90.92	91.03	91.04	91.06	91.12	90.86	91.18	91.00	91.08	91.06
85	91.03	91.07	91.24	91.22	91.38	91.07	91.20	91.29	91.21	91.19
90	91.09	91.13	91.13	91.18	91.45	91.09	91.27	91.16	91.37	91.34
95	91.28	91.40	91.43	91.44	91.46	91.27	91.51	91.47	91.34	91.49
100	91.42	91.43	91.41	91.39	91.50	91.38	91.53	91.45	91.44	91.47

Table 4.9. The Average Total Benefit (Wheel Expansion Method) (% of Upper Bound)

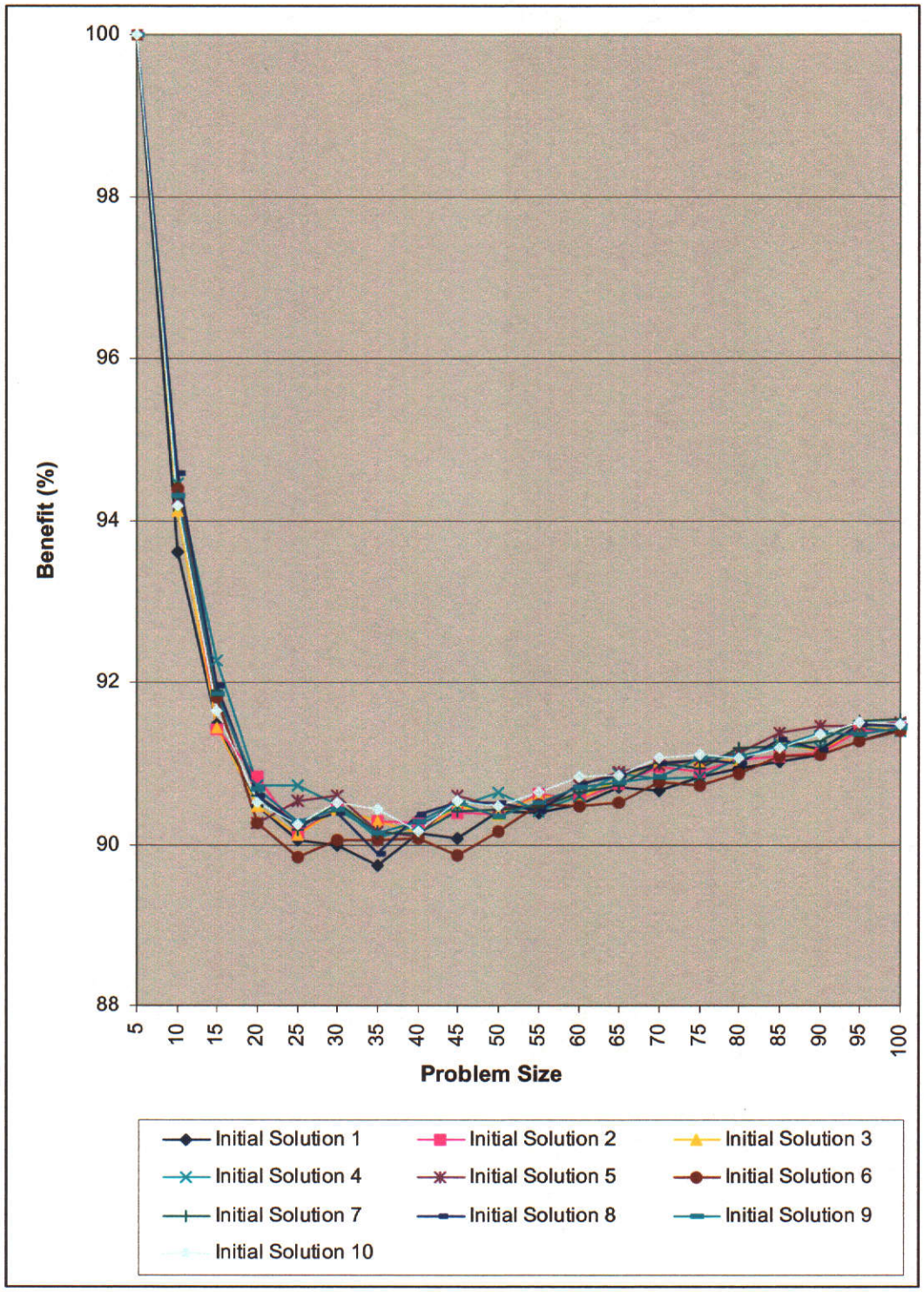


Figure 4.14. The Average Total Benefit (Wheel Expansion Method)

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	93.07	94.51	94.29	94.33	94.43	94.71	94.44	94.68	94.57	94.48
15	91.27	91.90	91.53	92.22	91.79	91.75	92.07	92.06	92.08	91.75
20	90.23	91.54	91.02	90.93	91.43	90.42	91.36	91.47	91.48	91.54
25	89.92	90.51	90.53	90.62	90.79	90.50	90.64	90.63	90.71	90.60
30	90.36	90.75	90.53	90.65	90.85	90.45	90.98	90.89	90.75	90.93
35	90.38	91.11	90.85	90.68	90.70	90.41	90.74	90.95	90.90	90.84
40	90.40	90.90	90.78	90.67	90.89	90.64	90.89	90.96	90.88	90.89
45	90.62	90.87	90.85	90.80	90.91	90.39	90.74	91.03	90.94	90.97
50	90.78	91.04	90.89	90.73	90.98	90.74	91.02	90.82	91.00	90.92
55	90.87	90.97	91.10	90.96	91.11	90.91	91.18	91.07	91.05	91.10
60	90.90	91.11	91.17	91.07	91.31	90.96	91.10	91.11	91.11	91.06
65	90.93	91.21	91.27	91.34	91.32	91.32	91.37	91.38	91.35	91.25
70	91.29	91.41	91.33	91.42	91.58	91.29	91.46	91.55	91.48	91.50
75	91.33	91.56	91.48	91.37	91.58	91.28	91.58	91.52	91.51	91.59
80	91.15	91.57	91.36	91.38	91.53	91.29	91.53	91.50	91.49	91.44
85	91.54	91.86	91.77	91.70	91.64	91.55	91.86	91.77	91.81	91.83
90	91.59	91.82	91.82	91.76	91.79	91.69	91.78	91.83	91.81	91.75
95	91.77	91.85	91.93	91.86	91.99	91.78	91.89	92.07	91.63	92.01
100	91.89	92.02	92.04	91.93	92.08	91.90	91.99	92.11	92.07	92.04

Table 4.10. The Average Total Benefit (Algorithm CK-1) (% of Upper Bound)

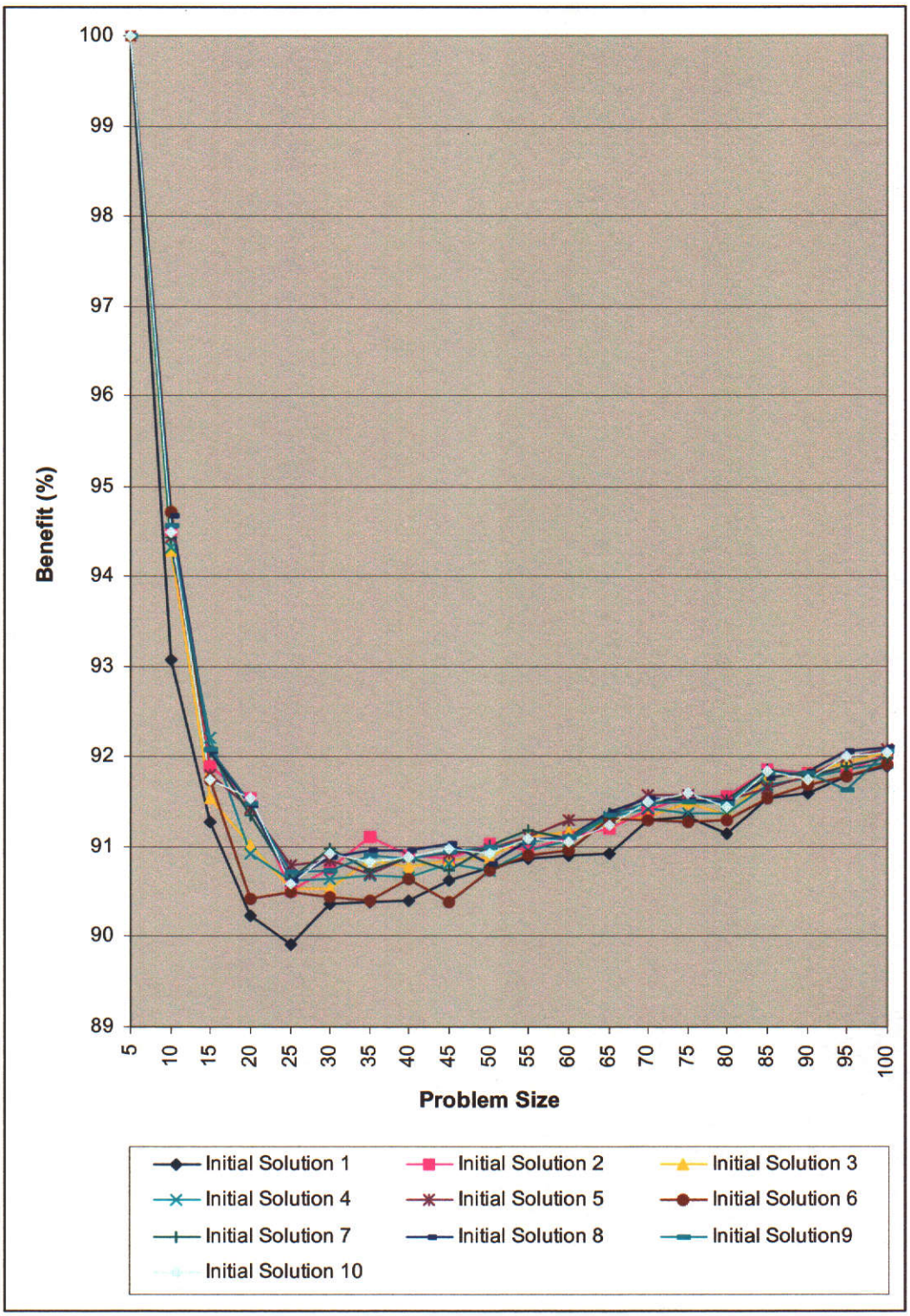


Figure 4.15. The Average Total Benefit (Algorithm CK-1)

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	93.17	94.89	94.95	94.74	95.11	94.60	94.94	95.14	95.13	95.15
15	91.70	91.80	91.96	92.50	92.73	91.88	92.51	92.78	92.13	92.62
20	90.58	91.77	91.50	91.17	91.85	91.16	91.59	91.77	91.76	91.69
25	90.49	91.18	91.12	91.04	91.46	90.72	91.17	91.24	90.83	91.35
30	90.65	91.17	91.04	91.13	91.49	90.92	91.47	91.36	91.03	91.45
35	90.74	91.48	91.44	91.25	91.41	90.93	91.30	91.27	91.05	91.46
40	90.86	91.34	91.42	91.10	91.37	90.91	91.30	91.28	90.90	91.28
45	90.97	91.28	91.24	91.27	91.42	90.90	91.37	91.45	91.08	91.35
50	91.13	91.43	91.48	91.31	91.53	91.08	91.58	91.63	91.22	91.66
55	91.26	91.43	91.60	91.35	91.53	91.25	91.65	91.61	91.18	91.57
60	91.39	91.69	91.64	91.50	91.77	91.37	91.75	91.77	91.31	91.75
65	91.35	91.70	91.76	91.69	91.95	91.39	91.72	91.76	91.53	91.87
70	91.65	91.98	91.92	91.89	91.94	91.64	91.93	91.96	91.56	91.98
75	91.72	92.04	91.92	91.87	92.02	91.82	91.96	92.10	92.05	92.07
80	91.68	91.98	91.97	91.93	92.02	91.93	92.10	92.09	92.10	92.03
85	91.95	92.34	92.20	92.14	92.32	91.89	92.39	92.29	92.35	92.27
90	92.06	92.26	92.22	92.14	92.35	92.04	92.27	92.28	92.27	92.37
95	92.25	92.35	92.44	92.28	92.44	92.20	92.52	92.47	92.20	92.54
100	92.23	92.51	92.52	92.40	92.51	92.32	92.48	92.46	92.13	92.43

Table 4.11. The Average Total Benefit (Algorithm CK-2) (% of Upper Bound)

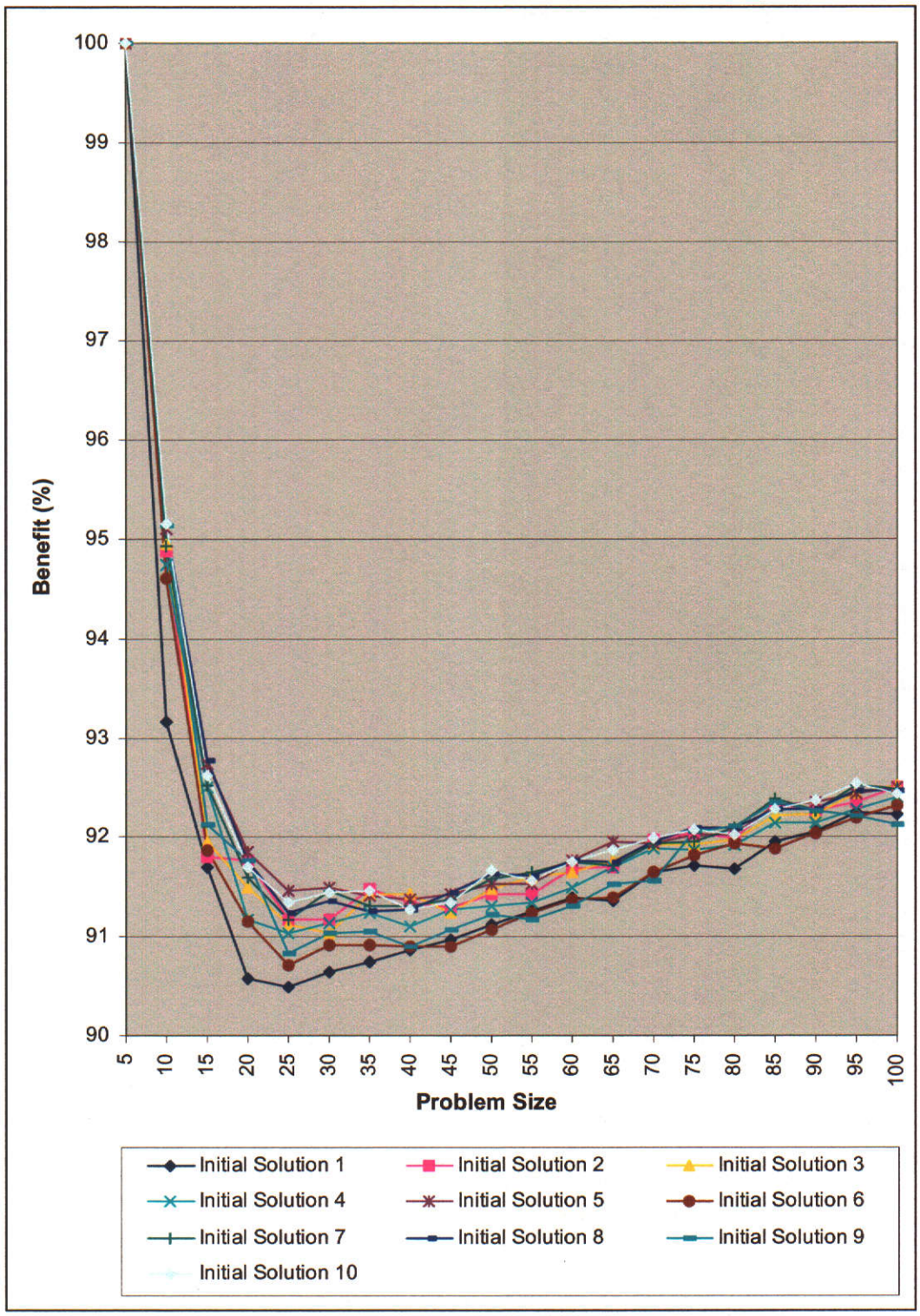


Figure 4.16. The Average Total Benefit (Algorithm CK-2)

n	Initial Solution Types									
	1	2	3	4	5	6	7	8	9	10
5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	93.17	94.89	94.95	94.74	95.11	94.60	94.94	95.14	95.13	95.15
15	91.70	91.80	91.96	92.50	92.73	91.88	92.51	92.78	92.13	92.62
20	90.58	91.77	91.50	91.17	91.85	91.16	91.59	91.77	91.76	91.69
25	90.49	91.18	91.12	91.04	91.46	90.72	91.17	91.24	90.83	91.35
30	90.65	91.17	91.04	91.13	91.49	90.92	91.47	91.36	91.03	91.45
35	90.74	91.48	91.44	91.25	91.41	90.93	91.30	91.27	91.05	91.46
40	90.86	91.34	91.42	91.10	91.37	90.91	91.30	91.28	90.90	91.28
45	90.97	91.28	91.24	91.27	91.42	90.90	91.37	91.45	91.08	91.35
50	91.13	91.43	91.48	91.31	91.53	91.08	91.58	91.63	91.22	91.66
55	91.26	91.43	91.60	91.35	91.53	91.25	91.65	91.61	91.18	91.57
60	91.39	91.69	91.64	91.50	91.77	91.37	91.75	91.77	91.31	91.75
65	91.35	91.70	91.76	91.69	91.95	91.39	91.72	91.76	91.53	91.87
70	91.65	91.98	91.92	91.89	91.94	91.64	91.93	91.96	91.56	91.98
75	91.72	92.04	91.92	91.87	92.02	91.82	91.96	92.10	92.05	92.07
80	91.68	91.98	91.97	91.93	92.02	91.93	92.10	92.09	92.10	92.03
85	91.95	92.34	92.20	92.14	92.32	91.89	92.39	92.29	92.35	92.27
90	92.06	92.26	92.22	92.14	92.35	92.04	92.27	92.28	92.27	92.37
95	92.25	92.35	92.44	92.28	92.44	92.20	92.52	92.47	92.20	92.54
100	92.23	92.51	92.52	92.40	92.51	92.32	92.48	92.46	92.13	92.43

Table 4.12. The Average Total Benefit (Algorithm CK-3) (% of Upper Bound)

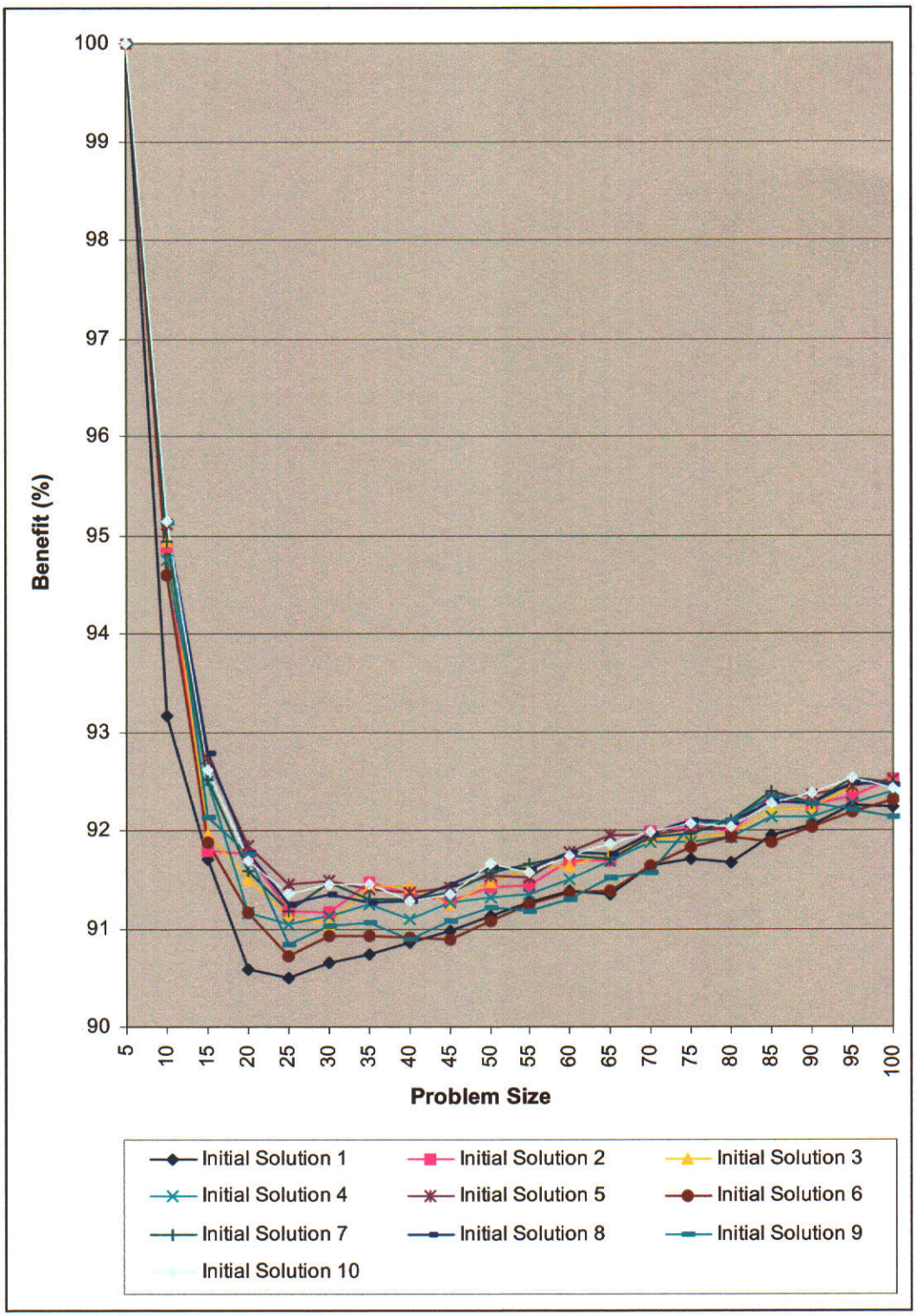


Figure 4.17. The Average Total Benefit (Algorithm CK-3)

Table 4.13 shows that the best initial solution for most algorithms is the Initial Solution 10, i.e. the best K_4.

Algorithms	Original Initial Solution	Best Initial Solutions
1. Deltahedron Method (R)	IS-10	IS-5
2. Kim-Kim Algorithm	IS-10	IS-10
3. Green-Al Hakim Algorithm	IS-5	IS-10
4. Leung's Constructive Heuristic	IS-10	IS-10
5. Wheel Expansion Method	IS-9	IS-5, IS-10
6. Algorithm CK-1	IS-10	IS-8
7. Algorithm CK-2	IS-10	IS-5, IS-10
8. Algorithm CK-3	IS-10	IS-10

Table 4.13. The Best Initial Solutions.

It is interesting to see that the Deltahedron Method (R-Construction) produces higher-weighted final solutions when Initial Solution 5 (IS-5) is implemented. From these results it is obvious that the Green-Al Hakim Algorithm shows that high-weight final solutions are obtained when IS-10 is used as its initial solution. Note that we do not construct a table for the Green-Al Hakim Algorithm as it has the same insertion operation as that of the Deltahedron Method. The only difference is its initial solution type which adopts a best triangle (initial solution type 5). Therefore, to see its performance for different types of initial solutions we refer to the table and the chart for the Deltahedron Method (see Table 4.6 and Figure 4.10).

It is noteworthy to see (from Table 4.10 to Table 4.12, and charts in Figure 4.11 to Figure 4.13) that Initial Solution 1 (IS-1) gives the worst final solutions to the Deltahedron Method, the Leung's Constructive Heuristic, and the Kim-Kim Algorithm. However, for the Wheel Expansion Algorithm, Algorithm CK-1, Algorithm CK-2 and Algorithm CK-3 it does not always contribute the lowest-weighted final solutions.

Table 4.5 and Table 4.6 (and Figure 4.11 and Figure 4.12) show that the

Kim-Kim Algorithm and the Leung's Constructive Heuristic give the best final solutions for Initial Solution 10 (IS-10). From Table 4.6 we can also see that the other good initial solution for the Leung's Constructive Heuristic is Initial Solution 5. Table 4.7 indicates Wheel Expansion Method, similar to Algorithm CK-1, CK-2, and CK-3, has best final solutions for Initial Solution 5 and 10. Note that in this experiment we do not include TESSA, as it is not competitive.

4.4 Discussions and Conclusions

We note that the best initial solution on the test problem does not always coincide with that proposed in the original description of the algorithms. For example, the Deltahedron Method (R-Construction) uses the best K_4 (i.e. Initial Solution 10) whilst the best in the test problems is Initial Solution 5.

Some heuristics are very sensitive to the structure of the Relativity Chart. Starting with different initial solutions, they may produce final solutions with different weight. This feature shows that these algorithms are not robust. From the experiments we conducted, it is obvious that the weight of final solutions are close to each other, showing that the different initial solution do not give variability of the weight of the final solutions, if edge removal is employed throughout vertex insertion operations.

CHAPTER 5
CONCLUDING REMARKS

This thesis has focused on the facility layout problem, which involves the determination of an optimal arrangement and configuration of facilities. The objective is to optimize profit, production cost, or material-handling cost. This problem has been studied for the past 40 years or so and a number of models and solution procedures have been proposed, including both exact methods and heuristic methods. Since the problem is NP-complete, a number of heuristics have been proposed with most of them based on graph theoretic concepts. Our main focus in this thesis is the graph theoretic approach. Graph theoretically, when the benefit of locating every pair of facilities is known, the problem can be modeled as that of finding, in a given edge-weighted graph, a maximum-weighted spanning subgraph that is planar. An advantage of the graph theoretic approach is that there is no need to use a planarity testing algorithm. A number of constructive heuristics have been proposed for generating a solution. Typically, these heuristics start with an initial solution that involves up to 4 vertices and other vertices are then inserted using well defined rules. The quality of the solution is influenced not only by the rules, but also by the initial starting solution.

The majority of the graph theoretic based literature heuristics build the solution by successive vertex insertions. Whilst some improvement procedures are sometimes applied to the final generated solution, none are implemented following a single insertion step. Consequently, once an edge is added during the insertion process it

usually remains in the final solution. Our new heuristics do allow single edge elimination at each step.

Our comparative analysis, based on 4200 randomly generated test problems from the uniform and normal distributions, demonstrates the improved performance characteristics of our algorithms. For a number of heuristics we have investigated various initial starting solutions. We presented the performance of each algorithm when various initial solutions are applied. We conclude this thesis by highlighting some important directions for further research on the area of the facility layout design, particularly on constructive graph theoretic based heuristics. These are suggested by our approach and the computational work has not been attempted because it is outside the time constraint of this project.

In our heuristics we were able to obtain improved results by allowing single edge elimination at each step. A natural extension is to consider multiple edge elimination at each step. Two-edge removal, for example, could be easily considered as follows. Consider the subgraph H depicted in Figure 5.1. The graph H-(b,d)-(b,e)

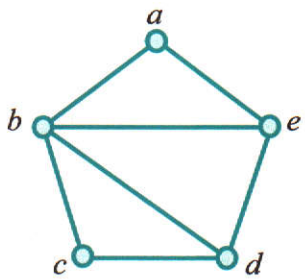


Figure 5.1. A subgraph of order 5.

is a 5-cycle. We could perform a single, double or triple vertex insertion to this subgraph and obtain the graphs displayed in Figure 5.2. We can clearly define the appropriate operation and extend our algorithm CK-3.

It may also be worthwhile to consider a 4-vertex insertion operation. We note, however, that the computational results indicate that heuristics which only employ single vertex insertion operations, double vertex insertion operations, or both, are time

efficient, compared to the heuristics which also implement triple vertex insertion operations. So the 4-vertex insertion operation may not be time efficient. Our experiments, however, show that if the inserted vertices are preselected (already specified before their insertions), then the processing time becomes efficient. It would be very interesting to further analyze and investigate the insertion operation using a number of pre-selected vertices, and whether it can save computing time and yet yield higher-weighted final solutions.

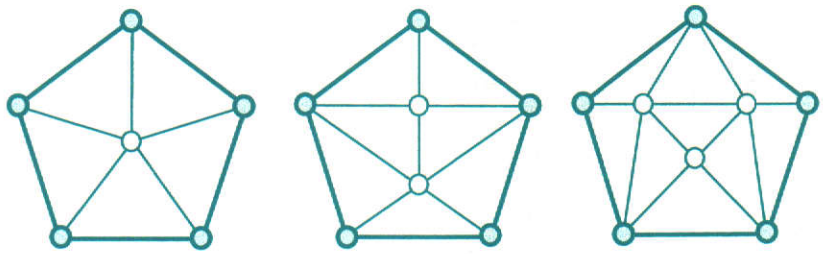


Figure 5.2. Vertex insertion operations into a 5-cycle.

Finally, we conclude this discussion by commenting on exact methods. We have noted in Chapter 2, some early work on branching algorithms (Branch and Bound and Branch and Cut), but very little attention has been given to these over the past 15 years or so. In fact, the latest use of the Branch and Bound method for solving FLDP dates back to 1980's, with the work of Burkard and Bonninger (1983), Burkard (1984), Lavalle and Roucairol (1985), and Kaku and Thompson (1986). Given the considerable progress that has been made over the past decade in solving large scale integer programming problems, using the Branch and Cut method, perhaps it is now time to revisit branching algorithms.

REFERENCES

Abdou, G., and Dutta, S. P. (1990). An integrated approach to the facilities layout using expert systems. *International Journal of Production Research*, **28(4)**, 685-708.

Al-Hakim, L. A. (1991). Two graph-theoretic procedures for an improved solution to the facilities layout problem. *International Journal of Production Research*, **8**, 1701-1718.

Al-Hakim, L. A. (1992). A modified procedure for converting a dual graph to a block layout. *International Journal of Production Research*, **30**, 2467-2476.

Al-Hakim, L. A. (1994). Discussion, A note on "On TESSA". *International Journal of Production Research*, **32**, 223-225.

Armour, G. E., and Buffa, E. S. (1963). A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, **9**, 294-309.

Bazaraa, M. S. (1975). Computerized layout design: A branch and bound approach. *AIIE Transactions* **7(4)**, 432-437.

Bazaraa, M. S., and Elshafei, A. N. (1979). An exact branch and bound procedure for quadratic assignment problems. *Naval Research Logistics Quarterly*, **26**, 109.

Bazaraa, M. S., and Sherali, M. D. (1980). Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Research Logistics Quarterly*, **27(1)**, 29-41.

Bazaraa, M. S., and Kirca, O. (1983). A branch-and-bound-based heuristic for solving the QAP. *Naval Research Logistics Quarterly*, **30**, 287-304.

Bondy, J. A., and Murty, U. S. R. (1976). *Graph theory with applications*. The Mac-Milan Press, London.

Boswell, S. G. (1992). TESSA-A new greedy heuristic for facilities layout planning. *International Journal of Production Research*, **30**, 1957-1968.

Bozer, Y. A., Meller, R. D., and Erlebacher, S. J. (1994). An improvement-type layout algorithm for single and multiple floor facilities. *Management Science* **40(7)**, 918-932.

Buffa, E. S., Armour, G. C., and Vollman, T. E. (1964). Allocating facilities with CRAFT. *Harvard Business Review*, **42**, 136-159.

Burkard, R. E. (1973). Die störungsmethode zur lösung quadratisches zuordnungs probleme, *Operations Research Verfahren* 16, 84-108.

Burkard, R. E., and Stratman, K. H. (1978). Numerical Investigations on quadratic assignment problems. *Naval Research Logistics Quarterly*, **25**, 129-144.

Burkard, R. E. (1984). Some recent advances in quadratic assignment problems. *Mathematical Programming*, edited by R. W. Cottle *et al.* (Amsterdam: North Holland).

Burkard, R. E., and Bonninger, T. (1983). A heuristic for quadratic Boolean program with applications to quadratic assignment problems. *European Journal of Operational Research*, **13**, 374-386.

Burkard, R. E., and Rendl, F. (1984). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, **17**, 169-174.

Burkard, R. E. (1984). Quadratic Assignment Problems. *European Journal of Operational Research*, **15**, 283-289.

Caccetta, L. and Kusumah, Y. (1998). A new heuristic algorithm for the facility layout design, in *Optimization, Techniques and Applications, ICOTA '98, Proceeding* (L. Caccetta *et al.* Editors), Curtin University of Technology, 287-294.

Caccetta, L., and Kusumah, Y. (1999). Graph Theoretic Based Heuristics for the Facility Layout Design Problem, in *Proceeding OR (Operations Research) in the New Millennium*, Operational Research Society of New Zealand.

Caccetta, L., and Kusumah, Y. (2000). Computational Aspects of The Facility Layout Design Problem. *Journal of Nonlinear Analysis, Theory, Methods and Applications* (to appear).

Carrie, A. S. (1975). The layout of multi-product lines. *International Journal of Production Research*, **13**, 541-557.

Chartrand, G., and Lesniak, L. (1986). *Graphs and Digraphs*, Wadsworth, Inc., Belmont, California.

Chiang, W. C., and Kouvelis, P. (1996). An improved tabu heuristic for solving facility layout design problems. *International Journal of Production Research*, **34(9)**, 2565-2585.

Connolly, D.T. (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research*, **46**, 93-100.

Dyer, M. E., Foulds, L. R., and Frieze, A. M. (1985). Analysis of heuristic for finding a maximum weight planar subgraph. *European Journal of Operational Research*, **20(1)**, 102-114.

Eades, P., Foulds L. R, and Giffin, J. W. (1982). An efficient heuristic for identifying a maximum Weight Planar Subgraph. In *Lecture Notes in Mathematics No. 952* (Combinatorial Mathematics IX). Springer-Verlag, Berlin.

Eggleton, R. B., Al-Hakim, L. A., and Macdougall, J. (1990). Braced edges in plane triangulations. *Australasian Journal of Combinatorics,* **2**, 121-133.

Eggleton, R. B., and Al-Hakim, L. A. (1991). Maximal Planar Graphs and Diagonal Operations. *Australasian Journal of Combinatorics*, **3**, 93-110.

Faigle, U., and Kern, W. (1992). Some Convergence Results for probabilistic tabu search. *ORSA Journal on Computing*, **4**, 32-37.

Fisher, E. L., and Nof, S. Y. (1984). FADES: Knowledge-based facility design. *Proceedings, International Industrial Engineering* Conference (Chicago, IL: Institute of Industrial Engineers), 74.

Foulds, L. R., and Robinson, D. F. (1976). A Strategy for Solving the Plant Layout Problem. *Operational Research Quarterly,* **27**, 845-855.

Foulds, L. R., and Robinson, D. F. (1978). Graph theoretic heuristic for the plant layout problem. *International Journal of Production Research*, **16**, 27-37.

Foulds, L. R. (1983). Techniques for facility layout: deciding which pairs of activities should be adjacent. *Management Sciences*, **29**, 1414-1426.

Foulds, L. R., and Giffin, J. W. (1985). A graph theoretic heuristics for minimizing total transport cost in facilities layout. *International Journal of Production Research*, **23**, 1247-1257.

Foulds, L. R., Gibbons, P.B., and Giffin, J. W. (1985). Facilities Layout Adjacency Determination: An Experimental Comparison of Three Graph Theoretic Heuristics. *Operations Research,* **33**, 1091-1106.

Fox, B. (1993). Integrating and accelerating tabu search, simulated annealing and genetic algorithms, F. Glover, T. Taillard, M. Laguna, D. De Werra (Eds.). Tabu Search, *Annals of Operations Research*, Baltzar, Basel, 47-67.

Frieze, A. M., and Yadegar, J. (1983). On the quadratic assignment problem. *Discrete Applied Mathematics*, **5**, 89-98.

Fujiwara, O., Makjamroen, T., and Grutta, K. K. (1987). Ambulance Deployment Analysis: A Case Study of Bangkok. *European Journal of Operational Research*, **30**, 9-18.

Gavett, J. W., and Plyter, N. V. (1966). The optimal assignment of facilities to locations by branch and bound. *Operations Research*, **14**, 210-232.

Gendreau, M., Laporte, G., Semet, F. (1998). Solving an Ambulance Location Model by Tabu Search. *Location Science* **5(2)**, 75-88.

Giffin, J. W., Foulds, L. R., and Cameron, D. C. (1986). Drawing a Block Plan from a REL chart with graph theory and a microcomputer. *Computer Industrial Engineering*, **10**, 109-116.

Gilmore, P. C. (1962). Optimal and Suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics,* **10**, 305-313.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, **5**, 533-549.

Glover, F. (1989). Tabu Search, part 1. *ORSA Journal on Computing*, **1(3)**, 190-206.

Glover, F. (1992). Ejection Chains, Reference Structures and Alternating Path Methods for Travelling Salesman Problems. University of Colorado.

Golany, B., and Rosenblatt, M. J. (1989). A heuristic algorithm for the quadratic assignment formulation to the plant layout problems. *International Journal of Production Research*, **27(2)**, 293-308.

Goldberg, J. R., Dietrich, R., Cheng, J. M., Mitwasi, M. G., Valenzuela, T., and Criss, E. (1990). Validating and Applying a Model for Locating Emergency Medical Vehicles in Tucson, AZ (case study). *European Journal of Operational Research*, **49**, 308-324.

Golden, B. L., and Skiscim, C. C. (1986). Using simulated annealing to solve routing and location problems. *Naval Research Logistics Quarterly*, **33**, 261-279.

Green, R. H., and Al-Hakim, L. (1985). A heuristic for facility layout planning. *Omega*, **13**, 469.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. Talk presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri.

Hassan, M. M. D., and Hogg, G. L. (1987). A review of graph theory application to the facilities layout problem. *Omega*, **15(4)**, 291-300.

Hassan, M. M. D., and Hogg, G. L. (1989). On converting a dual graph into a block layout. *International Journal of Production Research*, **27**, 1149-1160.

Hassan, M. M. D., and Hogg, G. L. (1991). On constructing a block layout by graph theory. *International Journal of Production Research*, **29**, 1263-1278.

Heragu, S. S., and Alfa, A. S. (1992). Experimental analysis of simulated annealing based algorithms for the facility layout problems. *European Journal of Operational Research*, **57(2)**, 190-202.

Heragu, S. S. (1992). Recent models and techniques for solving the layout problem (Invited Review). *European Journal of Operational Research*, **57**, 136-144.

Hertz, A., Taillard, E., and Werra, D. (1997). *Tabu Search, Local Search in Combinatorial Optimization*, Editor E. Aarts and J.K. Lenstra., John Wiley & Sons, Ltd.

Hillier, F.S. (1963). Quantitative tools for plant layout analysis. *Journal of Industrial Engineering*, **14**, 33-40.

Hopcroft, J., and Tarjan, R. (1974). Efficient Planarity Testing. *Journal of the Association for Computing Machinery*, **21**, 549-568.

Irvine, S. A., and Rinsma-Melchert, I. (1997). A new approach to the block layout problem. *International Journal of Production Research*, **35(8)**, 2359-2376.

Johnson, D. S., Aragaon, C. R., McGeoch, L. A., and Schevon, C. (1989). Optimization by Simulated annealing: An experimental evaluation. Part I, Graph partitioning, *Operations Research*, **37(6)**, 865-892.

Kaku, B. K., and Thompson, G. L. (1986). An exact algorithm for the general quadratic assignment problem. *European Journal of Operational Research*, **23**, 382-390.

Kaku, B. K., Thompson, G. L., and Morton, T. E. (1991). A hybrid heuristic for the facilities layout problem. *Computers and Operations Research*, **18(3)**, 241-253.

Kaufmann, L., and Broeckx, F. (1978). An Algorithm for the quadratic assignment problem using Benders' decomposition. *European Journal of Operational Research*, **2**, 204.

Ketcham, M. G. (1992). A branch and bound approach to facility layout design for continuous flow manufacturing systems. *International Journal of Production Research*, **30(3)**, 573-597.

Kim, J. Y., and Kim, Y. D. (1995). Graph Theoretic Heuristics for Unequal-sezed Facility Layout Problems. *Omega, International Journal of Management Science*, **23**, 391-401.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing, *Science*, **220(4598)**, 671-680.

Koopmans, T. C., and Beckman, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, **25**, 53-76.

Kouvelis, P., Chiang, W. (1992). Simulated annealing procedure for single row layout problems in flexible manufacturing systems. *European Journal of Operational Research*, **57(2)**, 203-223.

Kouvelis, P., Chiang, W., and Fitzsimmons, J. (1992). Simulated annealing procedure for machine layout problems in the presence of zoning constraints. *European Journal of Operational Research*, **57(2)**, 203-223.

Kouvelis, P., and Kiran, A.S. (1990). The plant layout problem in automated manufacturing systems. *Annals of Operations Research*, **26**, 397-412.

Kumara, S.R.T., Kashyap, R.L., and Moodie, C.L. (1987). Expert System for industrial facilities layout planning and analysis. *Computers and Industrial Engineering*, **12**, 143.

Kumara, S. R. T., Kashyap, R. L., and Moodie, C. L. (1988). Application of Expert System and pattern recognition methodology to facilities layout planning. *International Journal of Production Research*, **26**, 905.

Kusiak, A., and Heragu, S. S. (1987). Invited Review. The Facility layout problem. *European Journal of Operational Research*, **29(3)**. 229-251.

Kusiak, A., and Heragu, S. S. (1988). Knowledge based system for machine layout. *Proceedings, International Industrial Engineering Conference* (Orlando, FL: Institute of Industrial Engineers), 159.

Land, A. H. (1963). A problem of assignment with inter related costs. *Operations Research Quarterly*, **14**, 185-198.

Lavalle, I., and Roucairol, C. (1985). Parallel branch and bound algorithms, presented at EURO VIII, Bologna, Italy.

Lawler, E. L. (1963) The quadratic assignment problem. *Management Science*, **9(4)**, 586-599.

Leskowsky, Z., Logan, L., and Vanelli, A. (1987). Group technology decision aids in expert system for plant layout. *Modern Production Management Systems*, edited by A. Kusiak (Amsterdam: North-Holland), 561.

Leung, J. (1992). A new graph-theoretic heuristic for the facility layout. *Management Science*, **38(4)**, 594-605.

Li, Y., and Pardalos, P. (1992). Generating quadratic assignment test problem with known optimal permutations. *Computational Optimization and Applications*, **1(2)**, 164-184.

Malakooti, B., and Tsurushima, A. (1989). An expert system using priorities for solving multiple-criteria facility layout problems. *International Journal of Production Research*, **27**, 793-808.

Meller, R. D., and Bozer, Y. A. (1996). A new simulated annealing algorithm for the facility layout problem. *International Journal of Production Research,* **34(6)**, 1675-1692.

Moore, J. M. (1976). Facilities design with graph theory and strings, *Omega, The International Journal of Management Science*, **4(2)**, 193-203.

Nugent, C.E., Vollmann, T.E., and Ruml, J. (1968). An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, **16**, 150-173.

Padberg, M., and Rinaldi, G. (1991). A branch and cut algorithm for the resolution of large scale travelling salesman problems. *SIAM Review*, **33(1)**, 60-100.

Pierce, J. F., and Crowston, W. B. (1971). Tree-search algorithms for quadratic assignment problems. *Naval Research Logistic Quarterly*, **18**, 1-36.

Repede, J. F., and Bernardo, J. J., (1994). Developing and Validating a Decision Support System for Locating Emergency Medical Vehicles in Louisville, Kentucky. *European Journal of Operational Research*, **75**, 567-581.

Rinsma, F., Giffin, J. W., and Robinson, D. F. (1990). Orthogonal floorplans from maximal planar graphs. *Environment and Planning B: Planning and Design*, **17**, 57-71.

Sahni, S., and Gonzalez, T. (1976). P-complete approximation problems. *Journal of the Association of Computing Machinery*, **23**, 55-565.

Seppanen, J., and Moore, J. M. (1970). Facilities Planning with graph theory. *Management Science*, **17(4)**, B242-B253.

Seppanen, J., and Moore, J. M. (1975). String processing algorithms for plant layout problems. *International Journal of Production Research*, **13**, 239-254.

Sirinaovakul, B., and Thajchayapong, P. (1991). An intelligent approach to computer aided facility layout. *Proceeding of 1991 International Conference on Industrial Electronics, Control and Instrumentation*, 1, (Kobe, Japan: IEEE/IES and SICE), 87-90).

Sirinaovakul, B., and Thajchayapong, P. (1994). A knowledge base to assist a heuristic search approach to facility layout. *International Journal of Production Research*, **32**, 141-160.

Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing* **2(1)**, 33-45.

Souilah, A. (1995). Theory and Methodology. Simulated Annealing for manufacturing systems layout design. *European Journal of Operational Research*, **82**, 592-614.

Steinberg, L. (1961). The backboard wiring problem: a placement algorithm. *SIAM Review*, **3**, 37-50.

Suresh, G. and Sahu, S. (1993). Multiobjective facility layout using simulated annealing. *International Journal of Production Economics*, **32**, 239-254.

Taillard, E. (1991). Robust Taboo search for the quadratic assignment. *Parallel Computing*, **17**, 443-455.

Tam, K. Y. (1991). A simulated annealing algorithm for allocating space to manufacturing cells. *International Journal of Production Research*, **30**, 63-87.

Tompkins, J. A., White, J. (1984). *Facilities Planning*, New York: John Wiley & Sons.

Wascher, G., and Merker, J. (1997). A comparative evaluation of heuristic for the adjacency problem in facility layout planning. *International Journal of Production Research*, **35(2)**, 447-466.

Watson, K. H., and Giffin, J. W. (1996). On the worst case performance of Tessa. *International Journal of Production Research*, **34(10)**, 2963-2966.

Watson, K. H., and Giffin, J. W. (1997). The Vertex Splitting Algorithm for facilities layout. *International Journal of Production Research*, **35(9)**, 2477-2492.

Werra, D., and Hertz, A. (1989). Tabu Search Techniques: a tutorial and an application to neural network. *OR Spektrum* 11, 131-141.

Wilhelm, M. R., and Ward, T. L. (1987). Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, **19**, 107-119.