2001

Simulated annealing heuristics for the dynamic facility layout problem

Saravanan Kuppusamy
West Virginia University

SIMULATED ANNEALING HEURISTICS FOR THE DYNAMIC FACILITY LAYOUT PROBLEM

Saravanan Kuppusamy

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Industrial Engineering

Alan R. McKendall, Jr., Ph.D., Chair
Wafik H. Iskander, Ph.D.
B. Gopalakrishnan, Ph.D.

Department of Industrial and Management Systems Engineering

Morgantown, West Virginia
2001

Keywords: Dynamic Facility layout problem, Simulated Annealing, Forecasting windows, pairwise exchange heuristic

ABSTRACT

SIMULATED ANNEALING HEURISTICS FOR THE DYNAMIC FACILITY LAYOUT PROBLEM

Saravanan Kuppusamy

Today's consumer market demands that manufacturers must be competitive. This requires the efficient operation of manufacturing plants and their ability to quickly respond to changes in product mix and demand. Studies show that material handling cost makes up between 20 and 50 percent of the total operating cost. Therefore, this thesis considers the problem of arranging and rearranging (when there are changes in product mix and demand) manufacturing facilities such that material handling and rearrangement costs are minimized. This problem is called the dynamic facility layout problem. In this thesis, three simulated annealing heuristics are presented for the dynamic facility layout problem. The first is the direct implementation of the simulated annealing algorithm. The second heuristic uses a reheating strategy within simulated annealing. The third heuristic combines the simulated annealing algorithm, time windows concept, and the backward pairwise exchange method. The performance of the heuristics was evaluated using two measures: solution quality and computational time. Results obtained show that the proposed heuristics are effective for the dynamic facility layout problem.

ACKNOWLEDGMENTS

I express my deepest thanks to my advisor, Dr. Alan McKendall, Jr., for his time and effort. Under his tutelage, I have learned (and I am still learning) to approach real world problems and discussions with him always served me as catalysts to learn more. His reviews and comments on my thesis helped to improve my writing skills. Thank you sir!

I would like to thank Dr. Wafik Iskander, for his comments and reviews on my thesis. I always enjoyed his teaching, and I learned a lot from his patience and penchant for perfection. Thank you sir!

I wish to thank Dr.Gopalakrishnan for his continuous support throughout this research. Thank you sir!

I would like to thank my friends Mr. Karthik Palaniappan, Mr. Niranjan Mysore and Mr. Vivek Srinivasan for their invaluable tips on C-programming. Thank you friends!

I enjoyed my stay in Morgantown and I wish to thank all denizens of this wonderful town.

DEDICATION

I would like to dedicate this thesis to my parents. Their support and blessings stood with me throughout my academic career.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The Facility Layout Problem

The facility layout problem is the determination of the most efficient arrangement of departments within a facility. The facility can be manufacturing plants, administrative office buildings, or service facilities. The efficient arrangement of resources (e.g., machines, departments, or workforce) within the facility results in a well-coordinated workflow between the resources. An efficient layout helps other operations, which are dependent on workflow, to perform well. For instance, in a manufacturing plant, an efficient layout coordinates the material flow between the machines such that the right amounts of materials is supplied to the machines at the right time in a manner that is safe for workers and avoids the accumulation of work-in-process inventory. It also avoids the over-utilization of the material handling systems and reduces the material handling costs. Thus, an efficient layout assists a company and contributes to the overall efficiency of operations. In this research, the layout of a manufacturing plant is studied, and its material handling cost is used to measure the efficiency of the layout. However, Francis *et al*. (1992) mentioned other objectives that can be considered, and they are given below:

1. Minimize overall production time.
2. Minimize investment in equipment.
3. Utilize the space available effectively.
4. Facilitate the manufacturing process.
5. Promote effective utilization of manpower.
6. Maintain flexibility of arrangement and operation.

7. Provide for employee convenience, safety and comfort.
8. Minimize variation in types of material handling equipment utilized.

Material handling cost is the most significant measure for determining the efficiency of a layout and is most often considered, since it represents 20 to 50% of the total operating cost and 15 to 70% of the total cost of manufacturing a product (Tompkins et al., 1996, p.138). The material handling cost is based on the flow of materials and the distances between departments. Also, material handling costs depend on the material handling system being used. In other words, the type and the design of the material handling system influence the layout of the facility and vice versa. The traditional way of thinking forces one to believe that material handling systems should be designed after finalizing the layout or vice versa. On the contrary, the layout and the handling system should be designed simultaneously. Meller and Gau (1996a) point out that there exists a lack of concurrent engineering in designing the material handling system with respect to the facility layout, because material handling decisions are made before the layout is designed. Lack of concurrent engineering causes incompatibility between the facility layout and material handling design. That is, when the decisions based on the facility layout and material handling design are taken sequentially (without concurrency), it may result in a good layout with incompatible material handling equipment. In this research, the problem of determining the arrangement of departments within a manufacturing plant with respect to minimizing material handling cost is considered, and it is assumed that the material handling equipment is known.

1.2 Some Factors that can affect the Facility Layout

Tompkins *et al*. (1996, p. 287) mention some of the material handling decisions that affect the layout of a facility, and the major ones are: material handling equipment, material handling path, and handling unit size (unit load size). The material handling equipment and the material-handling path are not mutually exclusive. For instance, conveyors are material handling equipment used in a manufacturing plant to move the materials between specific points, such that the material-handling path is fixed. On the other hand, material handling equipment like forklifts, trucks and Automated Guided Vehicles (AGVs) are used to transport materials between various points, whereas the material-handling path is not fixed. Therefore, the decision of determining what type of material handling system (equipment) should be used has a significant impact on the effectiveness and flexibility of the facility layout. For instance, using forklifts in a flow shop is not economical since the volume of workflow between the machines is very high. Therefore, using forklifts can increase both work in process inventory and material flow congestion.

Another factor, which affects the layout, is the unit load. The number of units handled at a time is called the unit load. By handling batches of material at a time, it reduces the number of trips needed to transfer the material, material handling costs, and damage to the material while transferring. Containers, pallets, tote boxes, and cartons are some of the unit load containers used in industry. The size and weight of the unit load containers can influence and be influenced by the material handling equipment, which in turn influences and is influenced by the layout. For example, a forklift cannot be used to lift a container which is heavier than its lifting capacity. Also, a roller conveyor cannot be

used to transfer a tote box which is dimensionally incompatible. Thus, the unit load size is closely related to the material handling equipment being used, which is in turn associated with the facility layout.

Apart from the material handling decisions, there are some other factors that can also affect the layout, and they are as follows: change in the product design, the addition or deletion of a product, changes in the production methods, replacing obsolete equipment, and the adoption of new safety standards. A change in the product design calls for changes in the processes or operations to be performed. This change may require minor alterations in the existing layout or it may result in an extensive re-layout. When a new product is added, which is not considerably different from those in production, it may require minor alterations of the existing layout. Otherwise, new machines may be brought in for production, and the layout has to be changed in order to accommodate the new machines. This is also the case when there are changes in the production methods. When deleting a product, the material flow between departments (machines) may decrease. This reduction in the flow requires the layout planner to re-evaluate the layout. The evaluation may or may not result in changing the existing layout. In the case of obsolete equipment, additional space may be required to move the equipment. After removing the equipment more space may be available, and the layout may be altered to use this additional space. The extent of the layout alteration depends on the number of machines moved and their sizes. Last, when new safety standards are adopted, it may increase the space available for worker movement. The increased space requirement may burden the layout, and it may lead to changing the existing layout.

Gradual changes over time display themselves in terms of production bottlenecks, material flow congestion, failure to meet schedules, unexplainable delays, and increased idle time (Francis *et al*, 1992, pg. 25). Furthermore, over a period of time, demand changes. When the changes are left unattended, this may cause either a burden on the machinery and workforce needed to meet demand or to operate efficiently or under utilization of both machinery and workforce. In the former case, increased material flow increases space required to move the material; thus, causing flow congestion on the shop floor and burdening the capacity of the material handling equipment. In the latter case, reduced material flow reduces the utilization of the material handling equipment, machinery and workforce. As mentioned in the previous paragraph, it may result in under-utilization of resources. Therefore, the existing layout should be re-evaluated.

The re-evaluation of the existing layout takes place when there is either an increase or decrease in the material flow. Factors that cause the increase/decrease in the material flow were discussed in the previous paragraphs. Anticipating the changes in the material flow is imperative because these changes may lead to an extensive re-layout, which incurs high rearrangement costs (shifting costs) of machines. In addition, material handling cost will also vary. The problem of rearranging the layout of a facility when the material flow changes such that material handling and rearrangement costs are minimized is called the dynamic facility layout problem. This problem is studied in this research and is discussed in section 1.5.

1.3 Classification of facility layout problems

The objective function, distance measure, layout representation, and nature of the flow data are used to classify layout problems. Typically, there are two types of

objectives considered in the literature: distance-based and adjacency-based. Objectives (1)-(5) mentioned in section 1.1 are distance-based objectives and (6)-(8) are adjacency based. Minimization of the material handling costs is a distance-based objective since it is based on the interdepartmental flows and distances between departments. An adjacency-based objective is based on departments closeness ratings. The closeness rating is a numerical value, which indicates the preference between two departments being adjacent. By maximizing the adjacency score between preferred departments, the objective is achieved. This research uses a distance-based objective (i.e., minimization of the material handling cost) to arrange departments within a facility.

When considering distance-based objectives, a distance measure and the locations of the departments are required to determine the distance between two departments. The location of a department can be defined by its centroid or P/D (pick-up and drop-off) points. Pick-up points are the points where the materials are loaded onto material handling equipment, and drop-off points are the points where the materials are delivered. The distance between P/D points is the actual distance that affects the material handling cost. The locations of P/D points are not known until the detailed layout is developed and the material handling system is known. This leads to the widely used centroid to centriod approximation. Geometrically, the centroid can be defined as the center of gravity or center of mass of an object. In the facility layout domain, the centroid of a department is the center of the department, and it represents both the P/D points. Furthermore, rectilinear and Euclidean are the most commonly used distance measures. Rectilinear distance between any two points (x_1, y_1) and (x_2, y_2) is given by $|x_1-x_2|+|y_1-y_2|$, and Euclidean distance between two points is defined by $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$. This

research uses the centriod to centriod rectilinear distance measure to determine the distance between two departments. However, any distance measure can be used to obtain distances between departments.

After selecting the distance measure, the representation of the solution to the facility layout problem needs to be addressed. There are two ways in which the facility layout can be represented: block layout and detailed layout. A block layout specifies the relative location and size of each department within a facility. In addition, the block layout can be represented in either a discrete or continuous fashion. A discrete block layout representation uses a collection of grids to represent the departments, and a continuous representation uses the centroids, areas (or perimeters) and the widths (and/or lengths) of the departments to specify the exact locations of the departments. A layout, which specifies P/D points, aisle structures, and the layout within each department is called the detailed layout. A detailed layout can be represented in a continuous fashion. However, this research focuses on developing discrete block layouts.

The flow data used for determining the layout classifies the layout problem into two categories: static and dynamic. If the flow data between the departments does not change over time, then the problem is defined as the static facility layout problem (SFLP). A review of the SFLP approaches can be found in Kusiak and Heragu (1987) and more recently in Meller and Gau (1996a). When the flow changes over time, then the problem is defined as the dynamic facility layout problem (DFLP). Rosenblatt (1986) was the first to define the DFLP. A review of the DFLP approaches can be found in Balakrishanan and Cheng (1998). Furthermore, the nature of the flow data can be characterized as deterministic or probabilistic. Deterministic flow data is fixed and

known with certainty. That is, during the planning horizon, material flow between the departments is known with certainty. When the flow data are not known with certainty, they can be represented as random variables. That is, the behavior of the flow data can be approximated by a probability distribution. In other words, the flow data are said to be probabilistic. Kouvelis *et al*. (1992) addressed the probabilistic nature of the flow data for the facility layout problem. The flow data for the facility layout problem defined in this research are deterministic and dynamic. The classification of the layout problems is summarized in figure 1.1.

This research considers the facility layout problem, which uses the distance-based objective: minimization of material handling costs. The distance between departments is measured from centroid to centroid in a rectilinear fashion. In other words, the relative locations of the departments (block layout) are determined such that material handling cost is minimized. The block layout is represented in a discrete fashion, and the flow data for the problem considered is dynamic and deterministic. This problem will be defined and referred to as the DFLP.

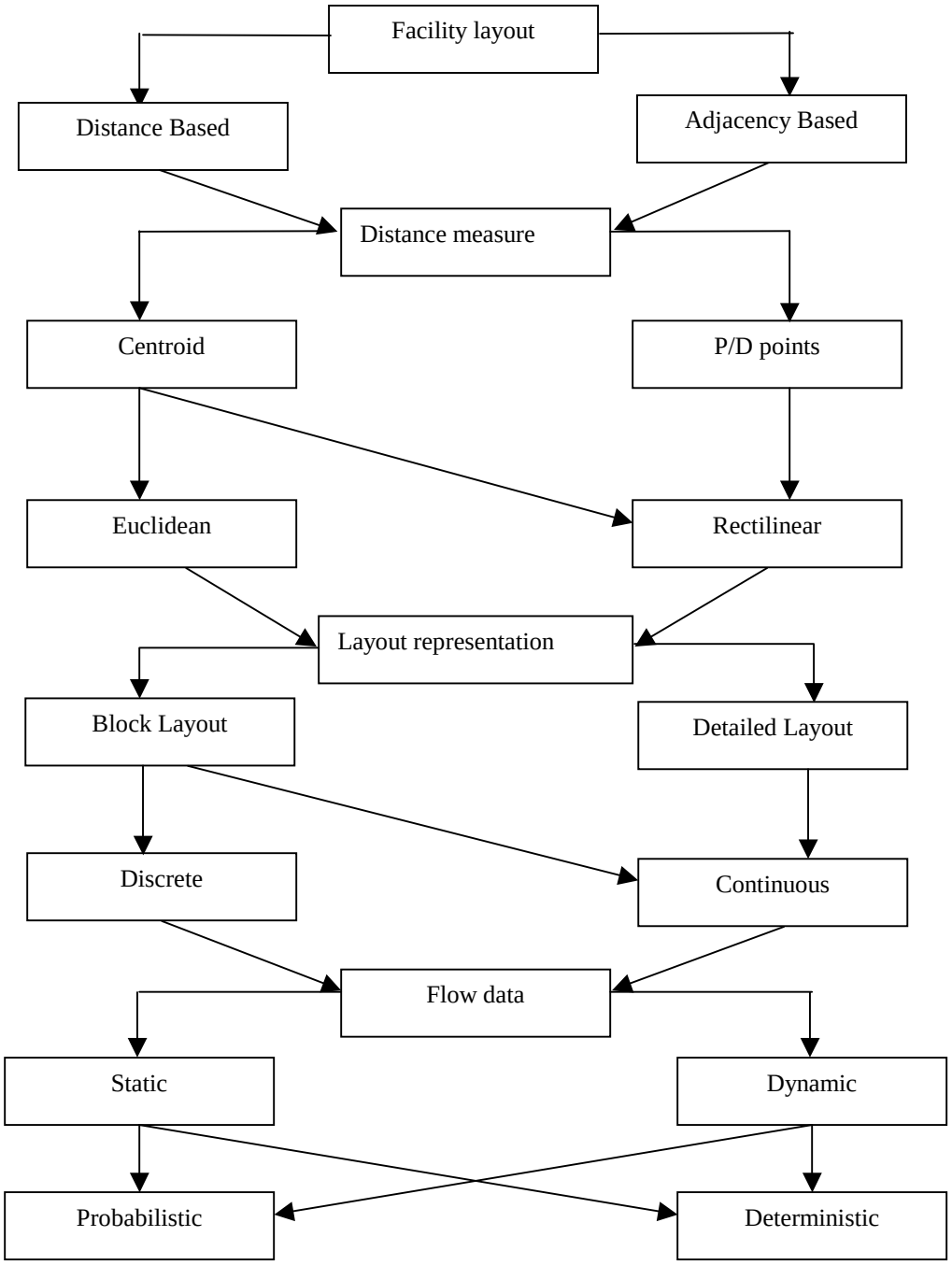


Figure 1.1: Facility layout classification

1.4 The Static Facility Layout Problem

For the SFLP, the flow between the departments does not change over time. The models presented in the literature for the SFLPs can be classified into discrete and continuous models, as mentioned in the previous section. That is, these models are classified based on the way in which the layout is represented. As previously stated, this research focuses on the discrete representation.

1.4.1 The Quadratic Assignment Problem

The Quadratic Assignment problem (QAP) model is a popular model and is used extensively in the literature. It has numerous real world applications such as the placement of electronic components on a printed circuit board, assignment of storage spaces on a computer disc, routing problems, assignment of departments to locations, to mention a few (Wilhelm and Ward, 1987). The QAP model is easy to comprehend, but it is computationally intractable. In other words, as the problem size increases, finding an optimal solution in reasonable time becomes extremely hard. The literature indicates that the largest problem for which an optimal solution has been found is for a 16-department problem (Urban, 1993). A review of the QAP based formulations and algorithms for solving SFLPs can be found in Kusiak and Heragu (1987).

The QAP model is used to assign facilities (departments) to locations such that the distance materials travel is minimized. The number of departments to be located and the number of locations are equal, and all the locations are of equal size. The location site diagram of a 6-department problem is given in figure 1.2.

1	2	3
4	5	6

Figure 1.2: Location site diagram

The distance between any two adjacent sites is one distance unit, and distance is measured from the centroid of one department to the centroid of another using the rectilinear distance measure.

The QAP formulation for the SFLP presented below is a similar version of the one presented in Koopmans and Beckman (1957).

$$\text{Minimize } Z = \sum_{i=1}^{N}\sum_{k=1}^{N}\sum_{j=1}^{N}\sum_{l=1}^{N} f_{ik}\, d_{jl}\, x_{ij}\, x_{kl} \qquad (1)$$

Subject to

$$\sum_{j=1}^{N} x_{ij} = 1,\ i = 1, \ldots, N \qquad (2)$$

$$\sum_{i=1}^{N} x_{ij} = 1,\ j = 1, \ldots, N \qquad (3)$$

$$x_{ij} = \{0,1\}\ \ i = 1, \ldots, N;\ \ j = 1, \ldots, N$$

where

N = Number of departments and locations.

f_{ik} = Flow between department i and department k.

d_{jl} = Distance between locations (sites) j and l.

$$x_{ij} = \begin{cases} 1 & \text{if department } i \text{ is assigned to location } j \\ 0 & \text{Otherwise} \end{cases}$$

The objective function (1) is used to minimize the distance materials travel. Constraint set (2) ensures that one location is assigned to each department, and constraint set (3) ensures that exactly one department is assigned to each location. The following section discusses the DFLP and the extension of the QAP model for the DFLP.

1.5 The Dynamic Facility Layout Problem

This research focuses on the Dynamic Facility Layout Problem (DFLP) with deterministic flow data. As discussed previously, when the flow data change over time, the facility layout problem is dynamic. The DFLP is based on the anticipated changes in flow that can occur in the future. The prospective future is divided into a number of time periods. In reality, the future can be divided into any number of periods and the time period may be defined in weeks, months or years. The flow data for each period are forecasted, and it is assumed that the flow data remain constant throughout the period. Therefore, the facility layout problem for each period can be considered as a SFLP and solved separately. However, after each period, a layout rearrangement may be necessary, and the rearrangement costs are not considered when solving the SFLP for each period seperately. As a result, the series of SFLPs has to be extended to the DFLP. Hence, a solution for the SFLP is a single layout, and a solution for the DFLP is a layout plan. See figure 1.3.

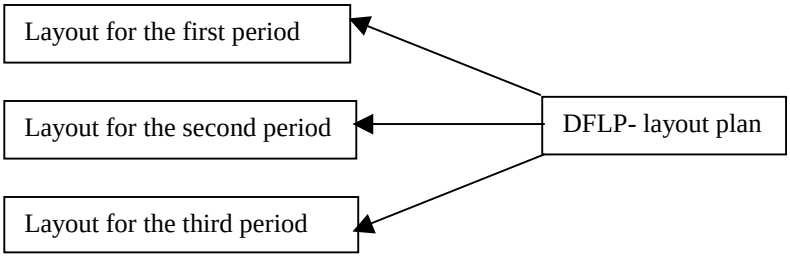


Figure 1.3: DFLP layout plan when the number of time periods is equal to 3

A layout plan for the DFLP is a series of layouts, and each layout is associated with a period. Therefore, the total cost of a layout plan consists of the sum of the material handling costs (flow costs) in all the periods and the rearrangement costs. The

rearrangement costs are incurred when the departments are rearranged in order to minimize the material handling costs. In a manufacturing environment, the rearrangement cost is incurred when moving machines from one place to another. The rearrangement of departments may cause production loss, and it may also necessitate the movement of machines adjacent to the machines moved. The rearrangement of machines may also require specialized labor and equipment. Thus, rearrangement cost comprises labor cost, equipment cost, and the cost of production loss (if necessary). Locally, rearrangement takes place in a period only if there is an improvement in the flow cost in that period, such that the improvement in the flow cost offsets the rearrangement costs. Globally, the total sum of rearrangement costs and flow costs are minimized. Therefore, a tradeoff exists between the flow costs and the rearrangement costs.

When compared to the SFLP, the DFLP is very recent. Rosenblatt (1986) was the first to address the DFLP. Like the SFLP, the DFLP is also computationally intractable. For this reason, most of the DFLP approaches in the literature use heuristics and the discrete representation of the layout. A review of the problem assumptions and solution approaches for DFLP can be found in Balakrishnan and Cheng (1998). The formulation of the DFLP is given below and is adapted from Balakrishnan *et al*. (1992).

$$\text{Minimize } Z = \sum_{t=2}^{T}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{l=1}^{N} A_{tijl}\, Y_{tijl} \;+\; \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} C_{tijkl}\, X_{tij}\, X_{tkl} \qquad (1)$$

Subject to

$$\sum_{j=1}^{N} X_{tij} = 1, \quad i = 1, \ldots, N \text{ and } t = 1, \ldots, T \qquad (2)$$

$$\sum_{i=1}^{N} X_{tij} = 1, \quad j = 1, \ldots, N \text{ and } t = 1, \ldots, T \qquad (3)$$

$$Y_{tijl} = X_{(t-1)ij} \; X_{til}, \quad i, j, l, = 1, \ldots, N, \; t = 2, \ldots, T \qquad (4)$$

$$X_{tij} = \{0,1\} \text{ for all } i, j, t$$

$$Y_{tijl} = \{0,1\} \text{ for all } i, j, l, t$$

Where

N = Number of departments and locations.

T = Number of time periods.

A_{tijl} = Cost of shifting department i from location j to l in period t (where $A_{tijj} = 0$).

C_{tijkl} = Cost of material flow between department i located at location (site) j and k located at l in period t.

$$X_{tij} = \begin{cases} 1 & \text{if department } i \text{ is assigned to location } j \text{ at period } t \\ 0 & \text{Otherwise} \end{cases}$$

$$Y_{tijl} = \begin{cases} 1 & \text{if department } i \text{ is shifted from location } j \text{ to } l \text{ at beginning of period } t \\ 0 & \text{Otherwise} \end{cases}$$

The objective function (1) is used to minimize the sum of the rearrangement and flow costs between the departments. Constraint set (2) ensures that each location is assigned only one department at each time period, and constraint set (3) ensures that exactly one department is assigned to each location at each time period. Constraint set (4) helps to add the rearrangement costs with the material flow cost if a department is shifted between locations in consecutive periods.

The above formulation is simple enough to capture the flow cost and rearrangement costs. The rearrangement costs are incurred when the departments are shifted. The layout rearrangement is necessary when the consumer demand and product

mix changes. Other factors may also lead to layout rearrangement as discussed at the end of section 1.2. All these changes need to be anticipated in order to accommodate those changes. It is important to note that the DFLP relies on the accuracy of the flow data when considering these changes.

In summary, the DFLP minimizes the sum of the layout rearrangement costs and the material handling costs (flow costs) over the planning horizon. Any changes in the flow of materials between departments may necessitate layout rearrangement. Layout rearrangement costs are incurred and these costs must be traded-off against the benefits (improved flow costs) derived from the modified layout. This research focuses on developing effective methodologies to solve the DFLP.

Chapter 2 reviews the SFLP and the DFLP literature. Chapter 3 presents the definition of the DFLP and the problem assumptions. Chapter 4 introduces the solution methodologies to solve the DFLP. The computational results are given in chapter 5, and chapter 6 concludes the research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The SFLP is a well-studied problem, and the literature dates back to almost five decades. Most recently, the focus has been on developing heuristics for this combinatorial optimization problem. On the other hand, the DFLP literature is very recent, and began in the mid-1980s. Since the foundation and methodologies for solving the DFLP developed from the SFLP literature, the SFLP literature is reviewed. Afterwards, the DFLP literature is reviewed.

2.2 The Static Facility Layout Problem

Generally, the methodologies for the Static Facility Layout Problem (SFLP) can be classified as optimal (exact) algorithms and heuristic (sub-optimal) algorithms. Branch and bound and cutting plane algorithms are used to solve the SFLP. Branch and bound algorithms were developed and implemented by Gilmore (1962), Lawler (1963), and Kaku and Thompson (1986), to mention a few. Bazaraa and Sherali (1980) developed a cutting plane algorithm based on Bender's partitioning scheme and later Burkard and Bonninger (1983) also developed cutting plane algorithms. For a review of the exact algorithms for the SFLP see Kusiak and Heragu (1987).

Montreuil (1990) presented a mixed integer programming formulation for the SFLP. Similarly, Heragu and Kusiak (1991) presented two new models for the SFLP: Linear continuous and linear mixed integer models. The main disadvantage of these exact algorithms is that they entail heavy computational requirements when applied even to small size problems.

In order to obtain good (near optimal) solutions in a reasonable amount of time, heuristic algorithms were developed. A heuristic can be defined as a well-defined set of steps for quickly identifying good quality solutions. The quality of a solution is defined by an evaluation criterion (e.g., minimize material handling cost), and the solution must satisfy the problem constraints. Basically, heuristic algorithms for the SFLP can be classified into four classes (Heragu, 1992):

- Graph Theoretic Algorithms
- Construction Algorithms
- Improvement Algorithms
- Hybrid Algorithms

The subsequent sections are devoted to heuristic algorithms, since this research proposes a heuristic methodology.

2.2.1 Graph Theoretic Algorithms

Seppanen and Moore (1970) introduced the graph theoretic concepts applied to the layout design. Hassan and Hogg (1989) outline the graph theoretic approach into three stages: In the first stage, an adjacency graph (also called a maximal planar weighted graph) is developed from the relationships between departments; the dual graph of the departmental relationships is constructed in the second stage; and the third stage converts the dual graph into a block layout. See Hassan and Hogg (1987) for the details of the graph theoretic procedure.

Graph theoretic algorithms can be found in Seppannen and Moore (1975), Foulds and Robinson (1976), Foulds and Robinson (1978), Carrie *et al*. (1978), Montreuil *et al*.

(1987), Goetschalckx (1992), and Eades *et al*. (1982). A review of the graph theoretic algorithms is presented in Hassan and Hogg (1987).

The graph theoretic procedure does not guarantee that departments having strong relationships will be adjacent (Hassan and Hogg, 1987). Added to the above limitation, it may produce irregular shape departments.

2.2.2 Construction Algorithms

A construction algorithm consists of successive selection and placement of departments until a layout design is achieved. Some of the construction algorithms are: *HC66* (Hillier and Connors, 1966), *ALDEP* (Seehof and Evans, 1967), *CORELAP* (Lee and Moore, 1967), *RMA Comp I* (Muther and McPherson, 1970), *MAT* (Edwards *et al*., 1970), *PLANET* (Deisenroth and Apple, 1972), *LSP* (Zoller and Adendorff, 1972), *Linear placement algorithm* (Neghabat, 1974), *FATE* (Block, 1978), *INLAYT* (O'brien and Abdel Barr, 1980), *SHAPE* (Hassan *et al*., 1986), *NLT* (Van Camp *et al*., 1991), and *QLAARP* (Banerjee *et al*., 1992). These algorithms can be used to provide initial solutions for improvement algorithms.

2.2.3 Improvement Algorithms

An improvement algorithm starts with an initial solution (existing layout). This existing layout is improved by exchanging the locations of a pair of departments. The exchange, which produces the best solution, is retained and the procedure continues until the solution cannot be improved any further or until a stopping criterion is reached. Hence, the solution quality of improvement algorithms often depends on the initial layout given. The above heuristic is called a pairwise exchange heuristic.

The following example is given to explain the pairwise exchange heuristic. Consider an assignment vector (i.e., initial layout) **a** = {2, 3, 5, 1, 6, 4} for a six department (N = 6) SFLP. The assignment vector gives the location of each department. That is, the first department is located at site 2, the second department is located at site 3, and the sixth department is located at site 4. The pairwise exchange heuristic starts with the above initial layout and evaluates the neighborhood to select the best pair of departments to exchange. The neighborhood can be defined as the set of solutions that can be obtained from a solution by exchanging the locations of any two departments. For instance, given the above initial layout (solution) there are fifteen solutions in its neighborhood. These fifteen solutions can be obtained by exchanging the following fifteen pairs of departments: (1,2), (1,3), (1,4), (1,5), (1,6), (2,3), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (4,6), (5,6). For example, the first pair of departments, (1,2), is selected for the exchange based on the fact that this pair improves the flow cost more than any of the other pairs. Therefore, after the exchange, the assignment vector is given as **a'** = {3, 2, 5, 1, 6, 4}. At this point, the pairwise exchange heuristic evaluates the neighborhood of **a'** (the current solution) and selects the best pair based on the improvement in the flow cost. This procedure continues until the solution cannot be improved any further. This strategy of the pairwise exchange heuristic is called the steepest descent strategy. It is obvious that the performance of the pairwise exchange heuristic depends on the initial layout. Also, it often converges to a local optimum. The greedy nature of the pairwise exchange heuristic impedes the heuristic from escaping the local optimal solution and from finding the global optimum.

When the locations of two departments are considered for an exchange, then the pairwise exchange heuristic is called a 2-opt exchange. The 3-opt exchange is similar to the 2-opt exchange except that it considers changing the position of three departments at a time. For a 6-department problem (N = 6), the 2-opt exchange heuristic has 15 neighboring solutions that are in the neighborhood of the current solution. In the case of the 3-opt exchange heuristics 40 neighboring solutions are in the neighborhood of the current solution. Since the implementation of the 3-opt is more complex, the 2-opt exchange heuristic is widely used in the literature. The proposed research uses the 2-opt (pairwise) exchange heuristic (imbedded within a simulated annealing heuristics) to solve the DFLP.

Armour and Buffa (1963) developed *CRAFT* (Computerized Relative Allocation of Facilities Technique) based on the 2-opt exchange heuristic. *CRAFT* is the first computerized technique used for the facility layout problem. *CRAFT* starts with the initial layout and evaluates all possible pairs of location exchanges. The location exchange, which results in the greatest cost reduction, is selected. This procedure continues until no further improvements can be made. *CRAFT* greatly depends on the initial layout (solution) provided, and the greedy nature of the pairwise exchange makes it susceptible to converge to a local optimum. Some more improvement algorithms for the SFLP are: *H63* (Hillier, 1963), *H63-66* (Hillier and Connors, 1966), *COL* (Vollman *et al*., 1968), *Sampling algorithms* (Nugent *et al*., 1968), *FRAT* (Khalil, 1973), and *COFAD* (Tompkins and Reed, 1976). All of these algorithms use the 2-opt (pairwise) exchange heuristic. Picone and Wilhelm (1984) developed the *Revised Hillier algorithm* whereas 3-opt and 4-opt exchanges are considered. *LOGIC* (Tam, 1992), *MULTIPLE* (Bozer and Meller,

1994), *FLEX-BAY* (Tate and Smith, 1995b), and *SABLE* (Meller and Bozer, 1996) are some of the other improvement algorithms for the SFLP.

As mentioned earlier, improvement algorithms rely on the initial solutions provided, and they employ a systematic procedure of location exchanges. The shortcomings of the improvement algorithms originate from the greedy nature of the systematic exchange procedure. “The greedy nature” of the procedure is exposed because only the location exchanges, which result in the greatest cost reduction, are accepted. Hence, the nature of the exchange procedure often impedes the algorithm from finding the global optimum and causes the algorithm to converge to a local optimum.

The recent developments of meta-heuristics like simulated annealing, tabu search, and genetic algorithms has greatly influenced the performance of improvement algorithms. These heuristics use a general search strategy like the pairwise exchange heuristic. However, they allow for uphill moves (i.e., accept non-improving exchanges or solutions) so as to escape from local optimal solutions such that the global optimal solution can be obtained.

2.2.3.1 Tabu Search

The tabu search (TS) heuristic can be described as a local search technique guided by the adaptive or the flexible memory structures (Pirlot, 1996). The adaptive memory of TS is implemented with reference to short term and long term components. Generally short-term memory prevents cycling (i.e., revisiting the same solution), and long-term memory is used for diversification (i.e., the diverse exploration of the solution space). Short-term memory records the moves that result in cycling, and those moves are forbidden (taboo) for a certain number of iterations. The number of iterations that the

moves are forbidden is called the tabu size. Additionally, an aspiration criterion is used to override the tabu restriction, if a restricted move gives the minimum cost ever in the search process. Long-term memory records the frequency of the moves and forces the algorithm to explore the solution space not yet visited. The way in which the neighbor is selected in the TS heuristic is quiet different from the simulated annealing (SA) heuristic. Unlike SA (random search strategy), all the members of the neighborhood are evaluated, and the best solution is selected. In summary, TS takes advantage of the history of the search process and embeds it into the search process (Chiang and Chiang, 1998).

Seminal papers presented by Glover (1989 and 1990) on TS spurred researchers to use this technique to solve a vast number of combinatorial optimization problems. Skorin-Kopov (1990) was the first to apply the TS heuristic to the QAP. Taillard (1991) applied TS to the QAP and used the parallelization methods to improve the speed of the TS heuristic. Skorin-Kopov (1994) refined the TS heuristic using target analysis and dynamic tabu size strategies. Kelly *et al*. (1994) applied various diversification strategies to find high quality solutions. Battiti and Tecchioli (1994) did a comparative study on SA and TS performance on the QAP. They presented a general discussion about fundamental differences between SA and TS and conducted numerical experiments for the QAP problem. The authors concluded that SA outperforms TS when a limited number of iterations are executed, and they also claim that the efficient memory usage of TS can obtain good solutions. Chiang and Kouvelis (1996) implemented TS with dynamic tabu list size strategies, intensification criteria and diversification strategies involving a penalty function (long-term memory structure).

2.2.3.2 Genetic Algorithms

The Genetic algorithm (GA) imitates the process of evolution. Each feasible solution is treated as an individual, and the fitness of an individual is measured by the cost function. A population is equivalent to a set of solutions. Two of the solutions (parents) are selected and subjected to breed. A crossover operator makes this possible. The healthy (having good fitness) offspring replaces the weaker parent. The mutation operator is used to improve the solution space by diversifying it. The algorithm continues until a predetermined number of generations is reached.

The following authors used GA to solve the QAP. Fluerent and Ferland (1994) implemented GA, which uses local search methods to improve the fitness of individuals. Tate and Smith (1995a) implemented genetic algorithm, and tested it on problem instances given by Nugent *et al.* (1968). Suresh *et al*. (1995) tested three standard crossover operators and proposed a new crossover operator to avoid infeasible solutions in the population. Ahuja *et al*. (2000) employed the greedy GA with new crossover schemes and an immigration scheme to promote diversity.

Generally, GA has not gained the acceptance of the operations research community as did TS and SA. The reason for this is often times it may generate infeasible solutions (Bean, 1994). More often then not, SA and TS perform better than GA.

2.2.3.3 Simulated Annealing

Until the seminal paper presented by Kirkpatrick *et al.* (1983) on the Simulated Annealing (SA) algorithm, improvement heuristics often times could not escape from converging to poor quality solutions or poor local optima. One example of this type of

heuristic is the pairwise exchange heuristic, as mentioned previously. However, SA overcame the greedy nature of the pairwise exchange heuristic by imbedding probabilistic features.

In statistical mechanics, the annealing process can be viewed as the gradual decrease in the temperature of a heated solid until the solid reaches the ground state. In other words, when the solid attains the ground state, it is said to be "frozen." Each state or configuration of a solid is defined by the set of arrangements of its atoms. The ground state is the lowest energy of all. The temperature should be reduced continuously and carefully so that at each temperature level thermal equilibrium is achieved. Otherwise, the final state of the solid will have many defects because of locally optimal structures (e.g., the structure of the glass, whereas crystal structure is a globally optimal one). The process, which gives locally optimal structures, is called rapid quenching.

The concept of statistical mechanics is summarized and compared with optimization problems. The following analogy in figure 2.1 is adapted from Johnson *et al*. (1989).

PHYSICAL SYSTEM	**OPTIMIZATION**
State	Feasible solution
Energy	Cost
Ground state	Optimal solution
Rapid quenching	Local search
Careful annealing	Simulated annealing

Figure 2.1: Annealing analogy

The SA procedure used to solve the SFLP is given in figure 2.2 and is modified from Chiang and Chiang (1998). The initial feasible configuration is the initial assignment of departments to locations. Often times this layout is generated randomly. However, some heuristic approaches use a construction heuristic to obtain an initial solution (or layout). The cost of an assignment is the total material handling cost, and a cooling schedule is a set of values assigned to the SA parameters. The SA parameters are the initial temperature, maximum number of iterations, the epoch length *e*, and a rule specifying how the temperature is reduced.

Given an initial layout, the cost of the initial layout is calculated. Next, a neighborhood solution is selected randomly, and then the cost of the neighbor is ascertained. If the cost of the neighbor is less than the current solution (i.e., $\Delta < 0$), the neighbor is accepted. If the cost of the neighbor is equal the current solution (i.e., $\Delta = 0$), randomly select another neighbor and repeat the process. Otherwise, the probability $P(\Delta) = e^{-\Delta / T}$ is calculated. A random number *x* is generated from the uniform distribution U(0,1). If the generated *x* is less than the P(Δ), then the move is accepted, otherwise it is rejected. These steps are repeated until a predetermined number *e* (epoch length) is reached. Then the temperature level is reduced, and the whole procedure is repeated until the maximum number of iterations is reached. The value of the initial temperature is chosen such that the probability of accepting uphill moves (i.e., $\Delta > 0$) is high. The idea behind this is to avoid getting trapped at a local minimum early in the search process. When the temperature has been reduced and becomes low, the probability of accepting uphill moves becomes low. The heuristic terminates once the specified number of iterations has been reached.

1. Obtain an initial feasible configuration (solution) *S*. Set *Min_cost* = Cost (*S*) and *Min_assign* = *S*;
2. Set the cooling parameters including the initial temperature, *T*, the cooling ratio *r*, the epoch length *e*, maximum number of iterations, *max*, and iteration counter, *iter* = 1.

3.1 For $1 \leq i \leq e$, do
- 3.1.1 Pick a random neighbor $S' \in N(S)$;
- 3.1.2 Let Δ = Cost (*S'*) - Cost (*S*);
- 3.1.3 If $\Delta < 0$,
  - set *S* = *S'*;
    - If (Cost (*S'*) < *Min_cost*),
      - set *Min_cost* = Cost (*S'*);
        - *Min_assign* = *S'*;
        - go to 3.1.1;
  - Else
    - If $\Delta > 0$,
      - Calculate $P(\Delta) = e^{-\Delta / T}$ (probability based on the Δ);
      - Generate random number $x \sim U(0,1)$;
      - If ($x < P(\Delta)$)
        - Set S = S', go to 3.1.1;
      - Else
        - go to 3.1.1;
    - Else
      - go to 3.1.1;

3.2 If (*iter* = = *max*)
- Return *Min_cost* and *Min_assign*;
- STOP;

Else
- Set *T* = *rT*;
- *iter* = *iter* + +;
- go to step 3.1;

Figure 2.2: SA algorithm

Burkard and Rendl (1984) were the first to apply the SA algorithm to the facility layout problem. Also, Wilhelm and Ward (1987) presented a SA algorithm for solving the facility layout problem. In this paper, the authors undertook an experimental evaluation for setting the parameters, and the equilibrium condition of the system was tested at each epoch based on the mean total costs of the assignments. Heragu and Alfa (1992) applied

the Hybrid Simulated Annealing (HAS) algorithm (uses a penalty algorithm to generate an initial solution and then improves the solution using SA) for the QAP and compared its results with their version of the SA algorithm, 2-opt algorithm, 3-opt algorithm, and Wilhelm and Ward (1987) version of SA. Results show that HSA and Heragu and Alfa (1992) version of SA performed better than Wilhelm and Ward (1987) SA algorithm. Chiang and Chiang (1998) presented the probabilistic tabu search and the hybrid tabu search applied to the QAP. Also, they implemented a SA algorithm and a TS heuristic to solve the QAP. Additionally, extensive computational experiments were conducted with all four heuristics: SA, TS, probabilistic TS, and hybrid TS. The authors concluded that the heuristics produce competitive and superior results. Last, Dowsland (1993) used the SA algorithm to solve packing problems. This article is mentioned here since the author used a reheating strategy, which is used in this research.

This research uses SA and SA with reheating as tools to solve the DFLP. Later in the methodologies section of this paper, the proposed simulated annealing heuristics applied to the DFLP are given and explained. Next, heuristics using a combination of heuristics to solve the SFLP are discussed.

2.2.4 Hybrid Algorithms

Bazaraa and Kirca (1983) classified hybrid algorithms, and define hybrid algorithms as algorithms that have the characteristics of both optimal and sub-optimal algorithms. As mentioned previously, optimal algorithms such as branch and bound and cutting plane methods are used to solve the SFLP. Any improvement, or graph theoretic algorithms are suboptimal algorithms. In general, hybrid algorithms are aimed at

capturing the characteristics of optimal algorithms, which produce optimal solutions, and the congenital quality of heuristics which solve problems quickly.

Hybrid algorithms can be found in Burkard and Stratman (1978), Bazaraa and Sherali (1980), and Bazarra and Kirca (1983). Kusiak and Heragu (1987) extended the above definition of hybrid algorithms to include algorithms that used construction and improvement methodologies. These algorithms can be found in Elshafei (1977), Scriabin and Vergin (1985), and Drezner (1980).

In the literature, the practice of combining available heuristics is very common. Yip and Pao (1994) combined genetic search and SA algorithms, and they use it to solve the QAP. Chiang and Chiang (1998) combined the TS with SA to add the probabilistic component to the TS technique, and they called the technique hybrid TS. For more hybrid techniques, see Pirlot (1996).

2.2.5 Static Facility Layout Extensions

Kusiak and Heragu (1987) reviewed the facility layout problem literature, which served as a foundation for this literature review for the SFLP. More recently, Meller and Gau (1996a) reviewed the facility layout problem literature. This literature review discussed the extensions of the facility layout problem (i.e., dynamic layout, stochastic layout, and multiple objective criteria).

Balakrishnan and Cheng (1998) reviewed the DFLP literature. More specifically, they reviewed the algorithms used to solve the DFLP. Most of the algorithms were heuristics except the Dynamic Programming algorithm proposed by Rosenblatt (1986). In the next section, these algorithms are discussed in detail.

2.3 The Dynamic Facility Layout Problem

2.3.1 Introduction

Based on the literature available, the solution approaches for the DFLP can be divided into exact (optimal) algorithms and heuristic (suboptimal) algorithms. Since the DFLP is computationally intractable, heuristic approaches are mostly presented for this problem. Most recently, Balakrishnan and Cheng (1998) reviewed the DFLP literature. As mentioned previously, most of the approaches reviewed were heuristics except the algorithm presented by Rosenblatt (1986). This is an exact approach, and it is computationally intractable when the problem size increases. Also, Rosenblatt (1986) used two approaches (to be discussed in subsequent section) to evade the computational difficulty. Lacksonen and Enscore (1993) applied exact algorithms like cutting planes and branch and bound algorithms for the DFLP. To tackle the computational complexity of solving large problems, the authors combined the above algorithms and presented heuristics to solve large problems. The computational burden associated with larger problems solidified the need for heuristics which can produce good solutions in reasonable time. Heuristics like the pairwise exchange heuristic (Urban, 1993 and Balakrishnan and Cheng, 2000b), GAs (Conway and Venkataraman, 1994 and Balakrishnan and Cheng, 2000a), TS (Kaku and Mazzola, 1997), and SA (Baykosaglu and Gindy, 2001) were proposed recently. Next, a brief discussion of the methodologies presented in the aforementioned articles is given.

2.3.2 Exact algorithms

Rosenblatt (1986) used dynamic programming to solve the DFLP. In the dynamic programming technique, each period is considered to be a “stage” and a static solution for

each period (stage) is called a “state”. The dynamic programming algorithm is solved using a recursive formula. A layout is selected for each period from a pre-specified set of layouts corresponding to the period such that the sum of the flowcosts and the rearrangements costs is minimized through the planning horizon. In order to find a globally optimum solution, the pre-specified set of layouts for each period must contain all possible layouts corresponding to the period. Thus, the dynamic programming approach is computationally intractable for larger size problems.

Rosenblatt (1986) considered two approaches to make the dynamic programming, a computationally viable technique for larger size problems. The first approach is based on the Ballou (1968) heuristic (used to solve the dynamic warehouse location problem), and the second approach considers generating the layouts randomly. Both approaches were tested on a small problem ($N = 6$, $T = 5$) and the results were reported. The results show that the first approach yields better solutions.

Lacksonen and Enscore (1993) presented five algorithms to solve the DFLP, and they are: CRAFT, Cutting planes, Branch and bound, dynamic programming, and cut trees. Cutting planes, branch and bound, and dynamic programming approaches are exact algorithms, and they demand high computational effort for larger size problems. In order to make them computationally tractable for larger size problems, they were combined with heuristics. The cutting plane algorithm was combined with an iterative exchange routine heuristic. The exchange routine was used to consider rearrangements, while the cutting plane portion was used to estimate the best assignment. Furthermore, a parallel branch and bound algorithm was modified to handle the multiple time periods. This algorithm terminates after 50,000 nodes are obtained and it stores the 25 most promising

nodes. Also, Rosenblatt's (1986) dynamic programming algorithm was used. The best n-states (layouts) are selected for each time period, in order to make the algorithm computationally tractable. The CRAFT algorithm, which handles the varying department sizes, was modified for the DFLP. Cut trees, one of the graphical layout techniques, was extended to model the DFLP. The authors developed a new set of data based on the factors which could affect the performance of the algorithm. All five algorithms were tested using this data. The DFLP test problems were for N = 6, 12, 20, and 30 departments and the time periods considered were T = 3 and 5. The cutting plane algorithm performed the best solution for all 32 problems. The exchange algorithm obtained the optimal solutions for N = 6, T = 3 test problems (4 instances). However, it's performance deteriorated as the problem size increases. The branch and bound algorithm was used to verify the optimal solutions for all N = 6, T = 3 test problems but was unable to obtain solutions in reasonable time for the larger test problems. The dynamic programming algorithm generally produced the worst solutions. The cut tree algorithm found optimal solutions for N = 6, T = 3 test problems but performed significantly worse as N and T increased.

2.3.3 Heuristic algorithms

2.3.3.1 Pairwise exchange heuristic

Urban (1993) used the pairwise exchange heuristic to solve the DFLP. This heuristic was completely different from the earlier approach presented by Rosenblatt (1986). Urban (1993) used the principle of forecasting windows for solving the DFLP. The idea behind this technique is to allow a layout to be used for any given block of periods, to avoid rearrangement costs. The flow data for one or more periods are

combined and used to determine a layout for the current period. In this technique an initial layout is given, only for the first period. For instance, when the time window = 1 and the current period is 1, this technique uses the flow data of the first period to improve the given initial layout. This improved layout (layout of the first period) is used as an initial layout for the second period along with the flow data of the second period, and an improved layout for the second period is obtained. This process is continued until the layout of the last period is improved. Similarly, when the time window = 2, and the current period is 1, the flow data of first and second periods are combined to improve the given initial layout. This improved layout (layout of the first period) is used as an initial layout for the second period along with the combined flow data of the second and third periods. Thus, the improved layout of the second period is obtained and used with the combined flow data of the third and fourth periods to obtain an improved layout for the third period. The process is continued until the layout of the last period is improved. Therefore, for *m* time windows there would be *m* solutions (layout plans) and the solution with the minimum cost is selected. This methodology is computationally tractable since it uses only a pairwise exchange heuristic. As a result, even when the problem size increases, a solution can be obtained in reasonable time. The author compared the proposed methodology with Ballou (1968) and the random heuristic presented by Rosenblatt (1986). Computational results indicates that Ballou (1968) method performs slightly better than the proposed heuristic for smaller size problems, but when the problem increases (more than 12 departments), Ballou's (1968) heuristic is not applicable due to the computational requirements. In all the instances, the proposed heuristic gave better results than the random heuristic.

Balakrishnan and Cheng (2000b) proposed two heuristics which improved Urban's (1993) pairwise exchange heuristic. The first heuristic uses the final solutions (*m* solutions) of Urban's (1993) heuristic and works backward to search for better solutions. The second heuristic combines Urban's (1993) heuristic with dynamic programming.

As stated earlier, the first heuristic produces "*m*" backward pass solutions based on the "*m*" solutions produced by Urban's (1993) heuristic. The backward pass pairwise exchange starts at *T*-1 period and continues on to the first period. In this heuristic, the rearrangement costs between periods *T*-1 and *T*, and the rearrangement costs between periods *T* and *T*+1 were considered. In this technique, each of the *m* solutions is at least as good as Urban's (1993) heuristic solutions. However, the flow data are not combined like in Urban's (1993) heuristic.

In the second heuristic, Urban's solutions were used as initial layouts for the dynamic programming algorithm. Similar to the first approach, Urban's (1993) solutions were embedded within dynamic programming, and the authors claim that dynamic programming would give solutions at least as good as Urban's (1993) heuristic. The proposed heuristics were tested on N = 6, 15, 30 departments and T = 5, 10 period problems. Results show that these heuristics improve the solutions given by Urban's (1993) heuristic.

2.3.3.2 Genetic Algorithms

Conway and Venkataramanan (1994) presented a solution technique based on GA to solve the DFLP. In this approach, feasible solutions, called strings, are generated randomly for the first generation. Thus, a string represents one entire layout plan for the planning horizon. These strings were selected with some probability increasing with their

fitness value. Then a random splicing position was generated and the strings were split into sub-strings. These sub-strings were then swapped. When infeasible solutions occurred as the result of swapping, additional swaps were made to restore feasibility. The strongest string (highest fitness value) is retained throughout generations. All of the other strings are replaced every generation. The proposed GA was tested on the data set presented in Rosenblatt (1986). The algorithm has a population size of 400 and 50 generations, and optimal solutions were obtained. Additionally, the algorithm was tested on a data set generated by the authors, and the best solutions were reported. Overall, this was an attempt to explore the suitability of GAs for solving the DFLP.

Balakrishnan and Cheng (2000a) refined the GA approach presented by Conway and Venkataramanan (1994). The authors employed the point-to-point crossover operator to crossbreed the two strong strings (based on the fitness function). Unlike all the members of every generation replaced (the approach followed by Conway and Venkataramanan, 1994), the proposed technique replaces only some individuals of the population. The proposed technique is called the " Nested loop Genetic algorithm". It consists of two loops: inner and outer loops. In the inner loop, crossover and mutation operators are used to breed the children (layouts). A feasibility test is applied to eliminate illegal children (infeasible solutions). The minimum cost legal child (feasible solution) layout replaces the maximum cost parent in the population. Also, a diversification strategy such as mutation is employed to drive the algorithm to search areas needed to be explored. Mutation may occur with minimum cost legal child layout. The mutated child replaces the maximum cost parent. Finally, the outer loop replaces some of the poor parents (having lower fitness values) of the population by layouts that were generated

randomly. By replacing the solutions, the outer loop prevents the inner loop from working on the same population for several generations. In other words, the outer loop actually diversifies the population. The termination of the algorithm is based on a threshold value. When the difference between the best child layouts in two successive generations is less than a threshold value, the algorithm is terminated. The proposed technique was tested on N = 6, 15, 30 departments and T = 5, 10 period problems. The proposed technique was compared with the technique proposed by Conway and Venkataramanan (1994). The former performed better than the latter for the 10 period test problems.

2.3.3.3 Tabu Search

Kaku and Mazzola (1997) applied the TS heuristic to the DFLP. In this paper, TS was applied in a two-stage search process that incorporates the diversification and intensification strategies. Three diversification strategies were discussed: random generation of initial layouts, construction of initial layouts using a construction heuristic and diversification using frequency-based tabu critieria. The second strategy was incorporated into the algorithm. The intensification strategy is implemented by the adaptive tabu list. In the adaptive tabu strategy, the tabu size reduces to half of its value when there is no significant improvement for pre-specified number of iterations.

During the first stage of the search procedure, a number of initial solutions are constructed using the diversification strategy, and a basic tabu search (with fixed tabu size) was applied on each constructed initial solution. At the end of the first stage, n-best solutions are selected for the intensification process in the second stage. In the second stage, a modified tabu search (adaptive tabu size) is applied to the initial solutions

corresponding to each of the n-best solutions. The proposed technique was tested on the data set provided by Lacksonen and Enscore (1993). The authors compared the results with the best-known solutions given by Lacksonen and Enscore (1993) and with the results obtained from Urban (1993) heuristic. The results show that the relative performance of the proposed technique was improved when the problem size is increased.

2.3.3.4 Simulated Annealing

Baykosaglu and Gindy (2001) were the first to apply SA to the DFLP. The approach taken by the authors is a straightforward implementation of SA to solve the DFLP. A randomly generated layout plan (i.e., layout for each period) is given as an initial solution. The SA algorithm improves the initial layout plan. As mentioned in section 2.2.3.3, the SA uses the pairwise exchange heuristic to determine and select a neighborhood solution. The pairwise exchange heuristic used in SA to solve SFLPs is modified for the DFLP. In the modified pairwise exchange heuristic, a neighboring solution is selected in two steps. In the first step, a period is selected, and in the second step, a pair of departments is determined randomly, from the period selected. After a period and a pair of departments are selected, the change in the total cost of the layout is calculated. This neighborhood solution is accepted based on the calculated change in the total cost. See section 2.2.3.3 for the SA procedure.

In Baykosaglu and Gindy (2001), the parameters such as initial temperature, final temperature, cooling rate, and maximum number of iterations are determined through an iterative procedure that is based on the upper and lower bounds of the solution of a given problem instance. Therefore, the authors performed several trial runs for each problem

instance to estimate the upper and lower bounds. The proposed SA was tested on the set of test problems (N = 6, 15, 30 for T = 5, 10) given by Balakrishnan and Cheng (2000a). The results obtained indicate that the proposed SA performs better than genetic algorithms.

As mentioned previously, the DFLP is computationally intractable, and it is evident that much of the research effort has been spent on developing heuristic approaches to solve the DFLP. This research proposes three heuristic approaches to solve the DFLP. These approaches are based on the SA algorithm. The first approach is a straightforward implementation of SA for the DFLP, the second uses the first approach with reheating, and the third approach is a combination of the forecasting windows technique (Urban, 1993), SA, and the backward pass pairwise exchange algorithm (Balakrishnan and Cheng, 2000b). Next, chapter 3 defines the DFLP and the problem assumptions.

CHAPTER 3

STATEMENT OF THE PROBLEM

3.1 Introduction

Today's consumer market demands that manufacturing companies need to be competitive. Every manufacturing company wants to prevail over the uncertainties of the demand. Moreover, when the demand changes with time, the burden to meet the demand increases. Being able to meet the demand determines the success of a manufacturing company. Naturally, companies have to find a way to meet demand that is cost effective and efficient. A company can look for improvements (cost savings) in planning, manufacturing, distributing, marketing, as well as other areas.

Within a manufacturing system, the layout of a facility can be explored to reduce costs. For instance, an inefficient layout will increase material handling costs, since the layout establishes a relationship between the departments based on the flow of materials between them. The material handling costs are linear functions of distance and flow volumes. Thus, an inefficient layout provides a good opportunity for cost saving (i.e., the opportunity to reduce material handling costs).

Material handling cost depends on the arrangement of the departments. More importantly, material handling cost associated with the layout has been estimated to be between 20% to 50% of the total operating expenses within manufacturing, and this cost can be reduced by at least 10% to 30% with effective facilities planning (Tompkins et al., 1996, p. 5). Since material handling cost is the function of distance and flow volumes, an

increase in flow volume will increase material handling cost. Thus, a layout, which is not ready to accommodate the changes in flow, will considerably lose its effectiveness.

When there is no change in the material flow over time, then the environment is said to be static. In today's consumer market, which is characterized by uncertainty, static environments are non-existent. The following section briefly discusses the static environment.

3.1.1 Static Environment

In the static environment, the flow of materials between the departments and the cost of handling materials per distance unit are used to evaluate a layout design. The flow of materials are assumed to be constant. However, in reality, the flow of materials between departments is not constant due to the volatile business environment.

Arguably, the facility layout in the static environment is designed in a reactive mode. In the reactive mode, future changes in the flow of materials are not taken into consideration, and the facility layout reacts to the change once it occurs. The static approach uses the long planning horizon and disregards changes in the flow (single period). That is, any changes in the flow during this planning horizon are left unattended. Therefore, an efficient layout at the start of the planning horizon may not be efficient throughout the time horizon.

3.1.2 Dynamic Environment

In a dynamic environment, the flow between departments changes with time. When the flow changes, the layout of the facility has to change since the changes in the flow with the current layout may increase material handling cost. Therefore, departments have to be rearranged in order to reduce material handling costs. When departments are

rearranged, the cost of shifting or rearranging the departments needs to be considered. The cost of physically moving machines (departments) from their existing location to their new location is called rearrangement cost. This rearrangement cost includes planning, dismantling, construction, movement and installation costs.

In the dynamic environment, the focus is to arrange the departments for each period with respect to minimizing the sum of material handling costs and rearrangement costs. Therefore, there is a tradeoff between minimizing material handling costs (flow costs) and rearrangement costs. In other words, if rearrangement costs exceeds the reduction in material handling costs when considering relayout, then the current layout is sufficient. However, if there is a reduction in material handling cost when considering re-layout, such that it exceeds rearrangement costs, the decision to relayout the facility is chosen. Since the flow data is extremely important for the relayout of facilities, the flow data need to be forecasted well in advance and need to accurate.

In the dynamic environment, the layout plan (layout for each period) is devised based on the multi-time period horizon. The flow data changes at discrete intervals. In this research, we assume that these changes are anticipated and are known in advance. The layout for each time period is designed based on the changes in the flow data and the rearrangement costs. The layout may be rearranged in one or more periods if the rearrangement cost does not counteract with the material handling costs in those periods. The output of the dynamic facility layout problem (DFLP) is a series of layouts, and each layout is associated with a time period. In summary, the objective of the DFLP is to obtain layouts for each time period such that the sum of rearrangement costs and flow costs (material handling costs) are minimized.

Lacksonen and Enscore (1993) state two limiting cases for the DFLP. First, when the rearrangement costs are much higher (prohibitive) than the flow costs, then any attempt to rearrange the departments (or facilities) would end up in a disaster. In this case, the flows for each time period can be summed up and solved as a SFLP. Second, when the converse is true, the DFLP can be solved as a series of independent static layout problems since the rearrangement costs are trivial. This research avoids these two extreme cases and concentrates on the transitional settings (rearrangement costs are neither trivial nor prohibitive) of the rearrangement costs.

3.2 Problem Statement

This research focuses on the DFLP where the flow data are deterministic and of course dynamic (flow data changes over the planning horizon). Changes in the flow are the results of many factors such as: the change in the design of an existing product; the elimination of products from a product line; the introduction of new products; replacements of existing production equipments; shorter life cycle products; and changes in the production quantities and associated production schedule. All these changes affect the flow of materials between departments. When the layout is rearranged to reduce the effect of changes in the flow of materials, material handling and rearrangement costs are incurred. In a manufacturing environment, the rearrangement cost is incurred when moving machines from one department to another. As mentioned earlier, there exists a trade-off between rearrangement cost and the improvement in the flow cost (material handling cost). The solution for the DFLP is a layout plan for the entire planning horizon. This layout plan is a series of layouts, and each layout is associated with a time period. The cost of a layout plan is the sum of the material handling and rearrangement costs.

At each time period T, a layout is one of the $N!$ (N factorial) arrangements of facilities, where N is the number of departments. A solution to the DFLP is one of the $(N!)^T$ layout plans available. Ironically, the problem stated here is very simple to explain but hard to solve optimally. For example, for $N = 6$ (six departments) and $T = 5$ (five periods), there are $(6!)^5 \cong 1.93 \times 10^{14}$ possible (feasible) arrangements. An extension of the QAP formulation is used to model this problem (see section 1.5).

3.3 Problem Assumptions

The assumptions for the DFLP are as follows:

- The type of layout is known. See figure 3.1 for an example of a 6 department problem. The layout is referred to as a 2 x 3 layout, and a department will be assigned to each location (or site) at each time period. The distance between sites and the flow between departments are used to obtain the objective function value (more specifically, material handling costs). Therefore, the type of layout must be known *a priori*.
- The distances between departments are determined *a priori*.
- Flow between departments is dynamic and deterministic.
- Departments and locations are of equal size.
- The assignment cost of a department to a location is ignored but can easily be considered.

Figure 3.1: Layout of Sites

3.4 Research Objectives

The major objectives of this research are as follows:

- To develop heuristics to obtain good solutions for large-size DFLPs.
    i. A straightforward implementation of SA algorithm.
    ii. A straightforward implementation of SA algorithm with reheating.
    iii. A combination of heuristics (SA algorithm with time windows and backward pass pairwise exchange heuristic).
- To evaluate the heuristics by comparing their results to results obtained from the best-known heuristics in the literature.

CHAPTER 4

METHODOLOGIES

4.1 Introduction

The DFLP is a computationally intractable problem. In other words, the computational effort to find the optimal solution for this problem grows exponentially with the size of the problem. It is known that exact algorithms entail high computational effort. Therefore, the need for heuristics to provide good solutions in reasonable time has increased. For the DFLP, there are several exact algorithms (Rosenblatt, 1986 and Lacksonen and Enscore, 1993), and the rest are heuristics (see section 2.3.1). More recently, SA was used to solve the DFLP (see Baykosaglu and Gindy, 2001). In this research, three heuristics based on SA are used to solve the DFLP, and they are as follows: a straightforward implementation of SA (SA I); SA I with reheating (SA II); and a SA algorithm combined with the pairwise exchange heuristic with time windows and a backward pairwise exchange heuristic (SA COMBO). These methodologies will be explained in the following manner: general idea (concepts), nomenclature, algorithm exposition, and illustration (numerical examples).

4.2 Simulated Annealing Algorithm (SA I)

This algorithm is a straightforward implementation of SA for the DFLP. This algorithm improves a given initial layout plan. The layout plan is a series of layouts (i.e., a layout is given for each period). Each period has its own flow data. Therefore, the input data for this algorithm consist of an initial layout plan, a series of flow data associated with each period, distance matrix (distances between locations), and rearrangement costs.

SA is a meta-heuristic, since it uses a general search strategy like the pairwise exchange heuristic within the SA algorithm. The way this meta-heuristic works can be divided into two loops: inner loop and outer loop. In the inner loop, a period is randomly selected. Next, a pair of departments for exchange is selected randomly. If the exchange yields a better solution than the initial solution (reduces material handling cost), then the exchange is made. In the outer loop, a decision to accept nonimproving exchanges is determined. This loop contains parameter initialization, stopping rules, and the probability assignment for accepting nonimprovement exchanges. The SA algorithm proposed, for the straightforward implementation, is adapted from Heragu and Alfa (1991) and Wilhelm and Ward (1987) and modified for the DFLP.

4.2.1 General Nomenclature

The following nomenclature is common for all of the proposed heuristics.

N = Number of departments.

T = Number of periods.

ax = $[a(1), a(2), \ldots, a(N)]$, a layout is represented by the n-vector; whereas the element $a(i)$ in the assignment vector ax denotes the site (or location) the department (or facility) i is located (or assigned).

layout _ plan = $[ax(1), ax(2), \ldots, ax(T)]$, where the element $ax(t)$ denotes the layout vector for the t^{th} period.

$\boldsymbol{ly^o}$ = The initial *layout_ plan*.

$\boldsymbol{ly}$ = The current *layout_plan* at any iteration.

$\boldsymbol{ly'}$ = The *layout_plan* considering a pairwise exchange.

Rep_cost = Total rearrangement cost for a *layout_plan*. That is, the sum of the rearrangement costs for all periods.

Flow_cost = The sum of the flow costs for all the periods.

Total_cost = Total cost of a layout plan; that is, the sum of the rearrangement costs (*Rep_cost*) and the material flow costs (*Flow_cost*).

Total_cost(***ly***) = Total cost of the current *layout_plan*.

Min_cost = The current best *Total_cost*.

Min_assign = The current best assignment of departments to locations in all periods (i.e., *layout_plan*).

Min_iter = Best *Total_cost* just before the last iteration.

Δ*flow* = Change in flow cost in a given period.

Δ*Rep* = Change in the total replacement cost (*Rep_cost*).

Δ*Total_cost* = Change in the *Total_cost* ; evidently, change in the *Total_cost* has two components Δ*flow* and Δ*Rep* and can be represented by (Δ*flow* + Δ*Rep*).

flow [*i*][*j*] = Material flow between departments *i* and *j* in a given period.

loc [*i*] = The location of department *i*.

d [*loc*[*i*]][*loc*[*j*]] = Distance between the locations of departments *i* and *j*.

4.2.2 Pairwise exchange heuristic

4.2.2.1 Concept

The pairwise exchange heuristic is an important component of the SA algorithm. In the DFLP, the pairwise exchange heuristic is implemented in two sequential steps:

1. The random selection of a period.

2. The random selection of two departments (*u* and *v*), in the selected period.

A swap (or exchange) between two departments in any given period is related to the flow cost in that period, and the rearrangement costs are calculated from rearrangement costs from the previous period to this period and from this period to the next period. This relation can be symbolically represented. See figure 4.1. In figure 4.1, if a swap occurred in period 2, then the flow cost of period 2 may change, and the replacement cost from periods 1 to 2, and from periods 2 to 3 may also change. An illustration is given in section 4.2.2.3

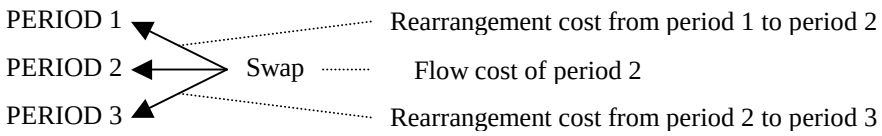


Figure 4.1: Relationship between flow and rearrangement costs

4.2.2.2 Formulas

4.2.2.2.1 Flow cost formulas

The flow cost of any layout can be calculated by using the following formulas:

$$\text{flow cost of a layout at any period} = \sum_{i=1}^{n}\sum_{i=1}^{n} flow[i][j]*d[loc[i]][loc[j]]$$

Generally, the above formula is suitable for both types of flow matrices (symmetric or asymmetric). In the case of a symmetric flow matrix, $flow[i][j] = flow[j][i]$, the above formula can be adjusted as follows:

$$\text{flow cost of a layout at any period} = \sum_{1\le i< j\le n} flow[i][j]*d[loc[i]][loc[j]]$$

4.2.2.2.2 Change in Flow (Δ *flow*) formulas

Once two departments are selected for interchange, the change in the flow cost is calculated. This computational formula is provided by Francis *et al*. (1992). When the flow matrix is symmetric, the following formula is used:

$$\Delta flow = \sum_{i=1}^{N} (flow[i][u] - flow[i][v]) * (d[loc[i]][loc[u]] - d[loc[i]][loc[v]]) -$$

$$2 * flow[u][v] * d[loc[u]][loc[v]]$$

When the flow matrix is asymmetric,

$$\Delta flow = \sum_{i=1}^{N} (flow[u][i] - flow[i][v] - flow[v][i] + flow[i][u]) * (d[loc[i]][loc[u]] -$$

$$d[loc[i]][loc[v]]) - 2 * d[loc[u]][loc[v]] * (w[u][v] + w[v][u])$$

where u and v are the departments selected for the exchange.

4.2.2.2.3 Change in Total Cost (Δ*Total_cost*) formulas

An exchange (swap) of departments in any period changes the *Total_cost* in two ways: changes the total replacement cost (Δ*Rep*) and changes the total flow cost (Δ*flow*). Since the flow cost in any period is not going to change except at the period where the swap takes place, the change in the *Total_cost* is Δ*Total_cost* = Δ*flow* + Δ*Rep*.

4.2.2.3 Illustration of calculating change in total cost

Consider the following *layout_plan*, (see figure 4.2), where T = 3, N = 6. The layout for the first period can be interpreted as the first department is located at site 2, the second department is located at site 3, the third department is located at site 4, and so on. For instance, period 2 is randomly selected and departments 5 and 6 (u = 5, and v = 6) are randomly selected for exchange. If the exchange reduces the total cost, then the location

of the departments are exchanged. The *layout_ plan* after the exchanges is given in figure 4.3.

2 3 4 6 1 5
1 2 3 4 5 6
4 5 6 1 3 2

Figure 4.2: *Layout_ plan* before the exchange

In figure 4.3, the layout shows that the replacement cost is reduced, since department 6 is assigned to location 5 during periods 1 and 2. However, department 5 is assigned to locations 1, 6, and 3 for periods 1, 2, and 3, respectively. Therefore, there is no reduction in replacement cost with respect to department 5. In this example, ΔRep = (Replacement cost of departments 5 and 6 in the previous *layout_plan*) – (Replacement cost of departments 5 and 6 in the current *layout_plan*).

Since there are reductions in the replacement cost of the current *layout_plan* (i.e., $\Delta Rep > 0$) and the flow cost ($\Delta flow > 0$), the total cost is reduced (i.e., $\Delta Total\_cost$ = ($\Delta flow$ in period 2) + $\Delta Rep > 0$).

2 3 4 6 1 5
1 2 3 4 6 5
4 5 6 1 3 2

Figure 4.3: *Layout_ plan* after the exchange (current layout)

4.2.3 SA I Nomenclature

The following are the parameters for the SA I algorithm and are also used for the SA II algorithm:

step = is a constant, which when multiplied by the temperature at a given iteration, determines the temperature used at the next iteration.

ini_temp = initial temperature.

iter_max = is a constant, that indicates the maximum number of iterations

executed without improvement in the *Total_cost*.

r = current iteration.

temp = current temperature.

= $(ini\_temp) * (step)^{(r-1)}$

NN = a constant which when multiplied by *N* defines the epoch length.

e = a constant that denotes the epoch length. The epoch length is defined as the number of accepted pairwise exchanges at each temperature (i.e., $e = NN*N$).

NS = a constant which when multiplied by *N* (number of departments) defines the maximum number of attempted pairwise exchanges in an epoch.

*NS*N* = a constant that denotes the maximum number of attempted pairwise exchanges in an epoch.

The SA parameters *step*, *ini_temp*, *iter_max*, *r*, and *temp* are collectively called the "annealing schedule." The annealing schedule is also called the cooling schedule, and it dictates how the temperature is reduced from start to finish. The equation that is used to calculate the current temperature (*temp*) is adapted from Wilhelm and Ward (1987). Initially *r* is set equal to 1, so that *temp* is equal to *ini_temp* at the first iteration. Before the temperature is reduced, a sufficient number of pairwise exchanges needs to be attempted. The number of accepted pairwise exchanges at a given temperature *temp* is called the epoch length. The parameters *NN* and *NS* are used to define the epoch length. The following are the counters, which are used to count the number of accepted pairwise exchanges, the

number of the attempted pairwise exchanges, and the number of non-improvement iterations, respectively:

ep = counter used to count the number of accepted pairwise exchanges in an epoch. The maximum limit of this counter is set equal to *NN***N*.

J = counter used to count the number of exchanges attempted at any temperature *temp*. The maximum limit of this counter is set equal to *NS***N*.

iter_no = counter used to count the number of iterations without improvement. The maximum limit of this counter is set equal to *iter_max.*

When the counters *ep* or *J* reaches the maximum limit, the temperature is reduced. Furthermore, when the counter *iter_no* reaches its maximum value, the algorithm is terminated.

4.2.4 SA I algorithm exposition

See figure 4.4 for the SA I algorithm. Generally, initial *layout_plans* are randomly generated. In this case, rather a simple *layout _plan* is given as an initial *layout_plan*. Basically, a layout for the first period (i.e., *ax*(1) = [1,2,3,4,5,6,7,8,9]) is used for all the periods. Note, that the layout for the first period is not generated randomly, but by simply assigning the first department to the first location, the second department to the second location, and so on.

Step 1:

The flow matrices for all the periods, distance matrix, and rearrangement costs are given as input. The simple initial *layout_ plan* ($\boldsymbol{ly}^{o}$) is assigned to $\boldsymbol{ly}$ (i.e., set $\boldsymbol{ly} = \boldsymbol{ly}^{o}$).

Step 2:

Total cost of the initial assignment, *Total_cost*(***ly***) is calculated; *Total_cost*(***ly***) is the sum of the rearrangement costs (*Rep_cost)* and the flow costs (*Flow_cost*).

Set *Min_cost* = *Total_cost*(***ly***);

Min_assign = ***ly***;

Min_iter = *Min_cost*,

r = 1 and *iter_no* = 0;

Step 3:

If *Min_iter* > *Min_cost*, then the number of non-improvement iterations (i.e., *iter_no*) is set equal to zero and the *Min_iter* is set equal to *Min_cost*. Otherwise, go to step 4.

Step 4:

In this step, *iter_no* is checked against the maximum non-improvement iterations allowed (i.e., *iter_max*). If counter *iter_no* reaches the *iter_max*, then stop, and *Min_cost* and *Min_assign* are returned. Otherwise, counter *iter_no* is incremented by one, and then go to step 5.

Step 5:

At every temperature reduction, the following parameters have to be initialized: *J* = 0 and *ep* = 1. Temperature is reduced according to the annealing schedule and is defined by $temp = (ini\_temp) * (step)^{(r-1)}$;

Step 6:

First, the period is selected randomly, and then two departments (*u* and *v)* are randomly selected in this period. The resulting assignment (i.e., with the exchange) is

denoted as $\boldsymbol{ly}'$. The change in the total cost is evaluated (i.e., $\Delta Total\_cost = Total\_cost(\boldsymbol{ly}) - Total\_cost(\boldsymbol{ly}')$). Last, *J* is incremented by one.

Step 7:

If $\Delta Total\_cost = 0$, then go to step 15 where the value of *J* is checked against the maximum limit *NS***N*. Otherwise, go to step 8.

Step 8:

If $\Delta Total\_cost > 0$, then go to step 11 where the selected pair of departments are accepted for the exchange. Otherwise, go to step 9 where the acceptance probability of the selected pair is determined.

Step 9:

If $\Delta Total\_cost < 0$, a random variable *x* is selected from a uniform distribution defined between 0 and 1 (i.e., U(0,1)). The acceptance probability of the selected pair is calculated based on the $\Delta Total\_cost$ and the temperature (i.e., *temp*). In other words,

$P(\Delta Total\_cost) = \exp(-\Delta Total\_cost / temp)$;

where,

$P(\Delta Total\_cost)$ = acceptance probability.

Step 10:

If $x < P(\Delta Total\_cost)$, go to step 11 where the selected pair of departments are accepted for the exchange. Otherwise, go to step 15 where the value of *J* is checked against the maximum limit *NS***N*.

Step 11:

In this step, the accepted pair of departments is exchanged and the counter (i.e., *ep*) for the number of accepted pairwise exchanges is incremented by one. The current

layout_plan (***ly***) is set equal to the *layout_ plan* considering the exchange (***ly***$'$) and the *Total_cost* is updated. In other words,

set *ep* = *ep* + 1;

ly = ***ly***$'$;

Total_cost(***ly***) = *Total_cost* - Δ*Total_cost*;

Go to step 12.

Step 12:

This step is to track the least cost assignment obtained considering the previous iterations. If *Min_cost* > *Total_cost*, then go to step 13. Otherwise, go to step 14.

Step 13:

If the condition stated in step 12 is satisfied, then the *Min_cost* is set equal to the *Total_cost* of the current *layout_plan* (***ly***) and the *Min_assign* is set equal to current *layout_plan*. Simply stated,

set *Min_cost* = *Total_cost* = *Total_cost*(***ly***);

Min_assign = ***ly***;

Go to Step 14.

Step 14

In this step, the value of the counter *ep* is checked against the epoch length, *e*. If both are equal, then this denotes the end of the epoch. The parameter *r* is incremented, and go to step 3. If the condition is not satisfied, then go to step 15.

Step 15

This step ensures that the algorithm attempted a predetermined number of exchanges (i.e., *NS***N*) at each temperature. The counter *J* is checked against its

maximum limit. If $J > NS*N$, then r is incremented. Then, go to step 3. Otherwise, go to step 6 where the algorithm selects a period and a pair of departments for exchange, and the algorithm continues in this fashion until *iter_no* equals *iter_max*.

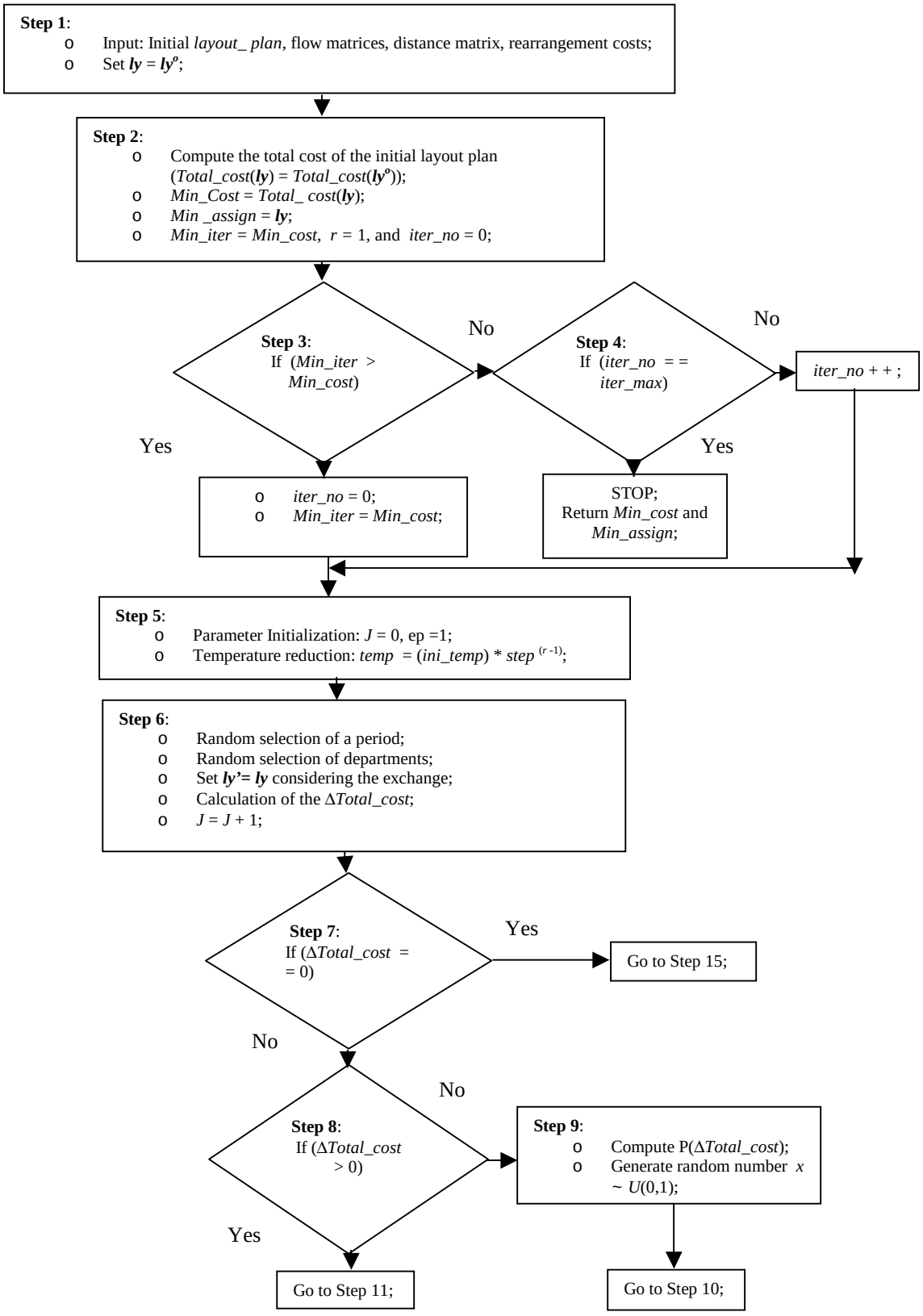
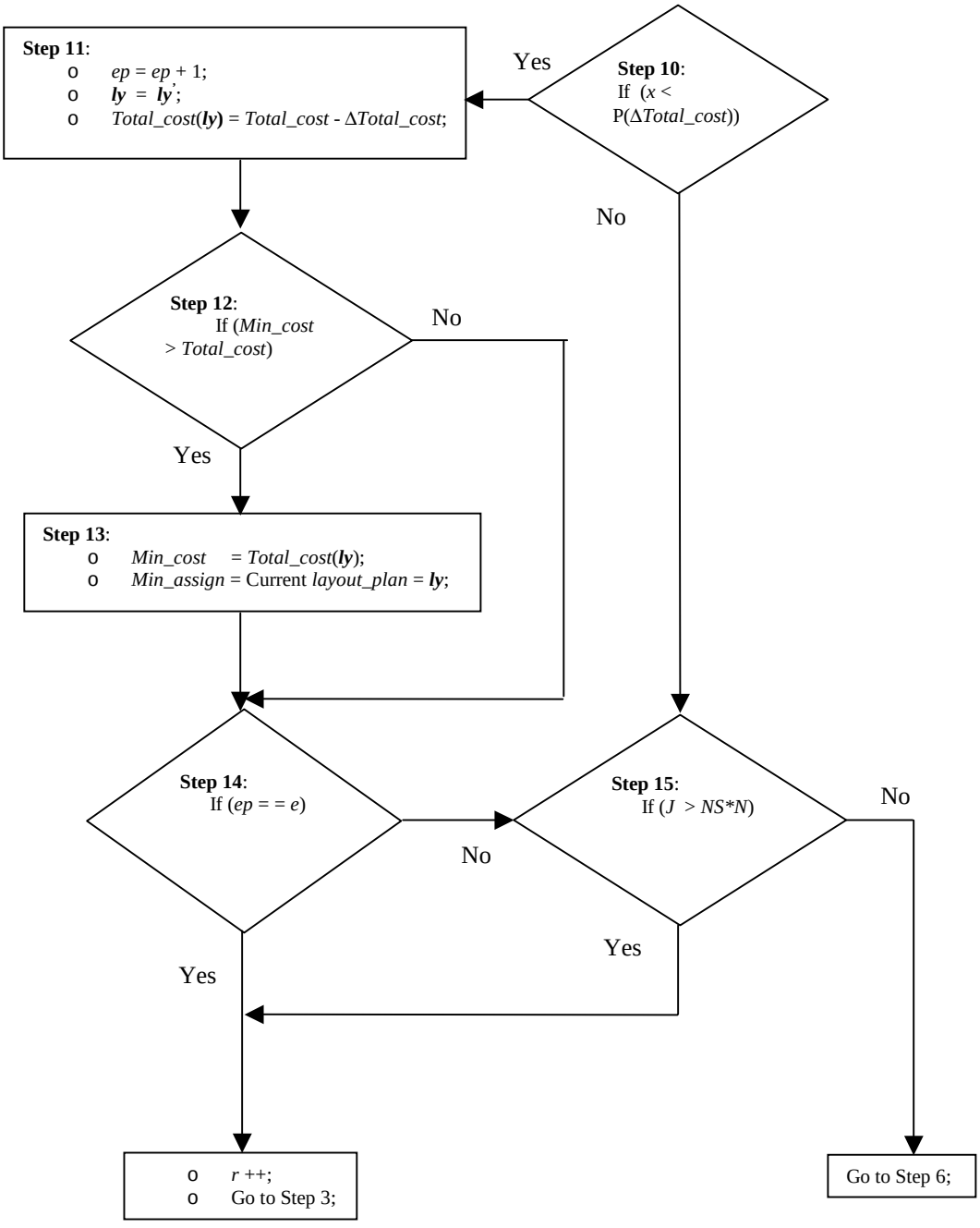


Figure 4.4: Simulated annealing algorithm (SA I)



Continue **Figure 4.4:** Simulated Annealing algorithm (SA I)

4.2.5 Illustration of the SA I algorithm

For illustration, a problem instance is taken from Conway and Venkataramanan (1994). This problem instance is a nine department problem (N = 9) with five periods (T = 5). The data for the problem instance is given in appendix A. This data contains the flow matrix associated with each period and a distance matrix (for a 3 x 3 layout). The parameters of the SA I are set at the following levels: *Step* = 0.99, *ini_temp* = 200000, epoch (e) = $NN*N$ = 90 (NN = constant = 10 and N = 9), maximum limit on J = $NS*N$ = 900 (NS = constant = 100), and *iter_max* = 1000. Note, the parameter setting is done for the purpose of illustration and is not based on the experimental evaluation. The level for the parameter *step* is chosen very close to 1 to ensure slow annealing. In other words, the temperature is reduced gradually. Moreover, slow annealing is possible only when a sufficient number of pairwise exchanges is allowed at each temperature. Thus, the maximum limit on the parameter J is set as high as possible. The level for the parameter e is set arbitrarily. Also, *ep* indicates the number of accepted pairwise exchanges at a given temperature. Whenever *ep* or J reaches the maximum limit (i.e., e and $NS*N*T$, respectively) the temperature is reduced. The initial temperature (*ini_temp*) is set equal to a high value (i.e., 200,000) to accept the nonimprovement pairwise exchanges initially.

Figure 4.5 represents the layout site (or locations) for the problem instance. The distances between locations are measured in rectilinear fashion, and the squares represent the sites (or locations), which are numbered.

1	2	3
4	5	6
7	8	9

Figure 4.5: 3 x 3 layout for problem instance

Step 1:

The initial *layout_plan* ($\boldsymbol{ly^o}$), flow matrices for all the periods, distance matrix and replacement costs for all the departments are given in appendix A. The initial *layout_plan* is as follows:

layout_plan = [1,2 3,4,5,6,7,8,9]

[1,2 3,4,5,6,7,8,9]

[1,2 3,4,5,6,7,8,9]

[1,2 3,4,5,6,7,8,9]

[1,2 3,4,5,6,7,8,9]

Note this *layout_plan* is not randomly generated. The first period layout is obtained by assigning the first department to the first location, the second department to the second location and so on. Also, the layout for the first period is used for all the periods. Thus, the *layout_plan* is simple and has no rearrangement costs, and the current *layout_plan* is set equal to initial *layout_plan* (i.e., $\boldsymbol{ly = ly^o}$).

Step 2:

The total cost of the *layout_plan* is calculated by using the formula discussed in section 4.2.2.2.1. Therefore, the following settings are made.

Total_cost(**ly**) = 674,479;

Min_cost = 674,479;

Min_assign = ***ly***;

Min_iter = *Min_cost* = 674,479;

r = 1 and *iter _no* = 0;

Step 3:

Since *Min_iter* = *Min_cost*, go to step 4.

Step 4:

Since *iter_no* = 0 and is less than *iter_max* = 500, *iter_no* is incremented (i.e., *iter_no* = 1). Go to step 5.

Step 5:

At every temperature reduction, the following parameters are initialized: *ep* = 1, and *J* = 0. The temperature is reduced according to the annealing schedule. Since *r* = 1, *temp* = (*ini_temp)** (*step*) * $^{(r-1)}$ = *ini_temp* = 200,000.

Steps 6, 7, and **8:**

Period = 3 is selected randomly. The first and sixth departments (*u* = 1, *v* = 6) are selected randomly for exchange. That is, ***ly'*** = [*ax*(1), *ax*(2), *ax*'(3), *ax*(4), *ax*(5)], where *ax*'(3) = [6,2,3,4,5,1,7,8,9]. This exchange improves the flow cost in the third period, and reduces the total flow costs (*Flow_cost*) by 1075 (i.e., Δ*flow* = 1075). However, this exchange increases the total replacement cost (*Rep_cost*) of ***ly'*** by 3424 (i.e., Δ*Rep* = - 3424). By exchanging the first and the sixth departments, it incurs rearrangement costs from periods 2 and 3 (i.e., 802 + 910 = 1712) and from periods 3 and 4 (i.e., 802 + 910 = 1712). Furthermore, the total rearrangement cost for the current *layout_plan* (i.e., ***ly***) is

zero. Thus, the change in the *Total_cost* is given by Δ*Total_cost* = 1075 – 3424 = - 2349. The counter *J* is incremented (i.e., *J* = 1). Since Δ*Total_cost* < 0, the conditions at steps 7 and 8 are not satisfied. Therefore, go to step 9.

Step 9:

In this step, the acceptance probability of the exchange is calculated based on the change in the objective function value.

P(Δ*Total_cost*) = exp(Δ*Total_cost*/*temp*)

= exp (-2349/200,000) = 0.9883.

Then, a random variable *x* = 0.8620 is generated from U(0,1). Go to step 10.

Step 10:

In this step, a decision is made whether to accept the exchange. Since *x* < P(Δ*Total_cost*), the exchange is accepted; else, it is rejected. Since *x* = 0.8620 < P(Δ*Total_cost*) = 0.9883, the exchange is accepted. Go to step 11.

Step 11:

The assignment considering the exchange is denoted as ***ly'***. After this exchange, *ax*(3) = [6,2,3,4,5,1,7,8,9]. The current assignment ***ly*** is set equal to ***ly***$^{'}$ (i.e., ***ly = ly***$^{'}$). The counter *ep* is incremented (i.e., *ep* = *ep* + 1 = 2), and the *Total_cost* is updated,

Total_cost = *Total_cost* - Δ*Total_cost* = 674,479 – (- 2349) = 676,828. Go to step 12.

Steps 12, 13:

As mentioned earlier, this step is to track the least *Total_cost* assignment. Since *Min_cost* (674,479) < *Total_cost* (676,828), step 13 is bypassed. Go to step 14.

Step 14:

At this point (i.e., *ep* = 2), *ep* is compared with the epoch length (*e* = 90). Since *ep* < *e*, go to step 15.

Step 15:

The value of the counter *J* is compared with its maximum limit which is 900 (i.e., *NS*N* = 900). Since *J* < 900, go to step 6. Therefore, the algorithm continues and begins another iteration at step 6. After several iterations and the number of attempted exchanges is 900 (i.e., *J* = *NS*N* = 900) at the current temperature, the outer loop counter is increased by 1 (i.e., *r* = 2) and the temperature (i.e., *temp*) is reduced. The algorithm is repeated as above at the new current temperature, and the algorithm continues in this fashion until the stopping criterion is met (i.e., when *iter_no = iter_max*).

4.3 SA with reheating (SA II)

The SA algorithm with reheating (SA II) is another one of the proposed heuristics. It is believed that reheating improves the quality of the solutions (Dowsland, 1993). In the SA algorithm, reheating can be considered at the stage where SA slowly converges to a local optimum. In other words, reheating should occur when the temperature is very low, and the probability of accepting up-hill moves is very low. Therefore, reheating increases the temperature enough so that up-hill moves may be accepted.

The following are the parameters specific to the SA II algorithm:

rehe_max = a constant that denotes the maximum number of non-improvement iterations that can be executed before reheating commences.

temp_inc = a constant that is deducted from the current iteration number *r*, when reheating commences.

In addition, to the counter *iter*_no, which is used to count the overall number of non-improvement iterations, *reheat* is a similar counter used within the reheating strategy:

reheat = counter used to count the consecutive number of iterations without improvement within the reheating strategy. The maximum limit of of this counter is set equal to *rehe_max*. If a solution obtained improves the *Total_cost*, then *reheat* is initialized to zero.

In figure 4.6, the reheating loop is inserted between steps 2 and 3 of the SA I algorithm. Initially, at step 2 the counter *reheat* is initialized to zero, along with the other counters *r* and *iter_no*. At the start of the algorithm, since the value of the counter *reheat* is not equal to *rehe_max,* go to step 3. As discussed in section 4.2.4 (SA I algorithm), the conditions at steps 3 and 4 will be evaluated. If the conditions are not satisfied, the counters *iter_no* and *reheat* will be incremented (i.e., *iter_no* = 1 and *reheat* = 1). At each non-improvement iteration, the counters *iter_no* and *reheat* are incremented. When the value of the counter *reheat* is equal to *rehe_max*, the reheating commences. In other words, the temperature (i.e., *temp*) is increased when *temp_inc* is subtracted from the current iteration number *r*. For instance, following are the levels of parameters when reheating commences: *step* = 0.99, *ini_temp* = 100, *r* = 500, *temp* ≅ 0.6636, *reheat* = 400, *rehe_max* = 400 and *temp_inc* = 200. Since *reheat* = *rehe_max* = 400, reheating commences, and the *temp_inc* is subtracted from *r* (i.e., *r* = *r* – *temp_inc* = 500 - 200 = 300). By substituting the value of *r* in the formula $temp = (ini\_temp) * (step)^{(r-1)}$, the *temp* is increased from 0.6636 to 4.9356 (i.e., $temp = 100 * 0.99^{(300-1)} \cong 4.9356$). After the temperature increases, the counter *reheat* is set equal to zero (i.e., *reheat* = 0). If the condition at step 3 is satisfied (i.e., there was an improvement in the *Total_cost* in the last

iteration), the counters *reheat* and *iter_no* are set equal to zero, and *Min_iter* is set equal to *Min_cost* (i.e., *reheat* = *iter_no* = 0 and *Min_iter* = *Min*_cost). If the condition at step 4 is true (i.e., *iter_no = iter_max*), the algorithm stops and returns minimum cost and assignment (i.e., *Min_cost* and *Min_assign*).

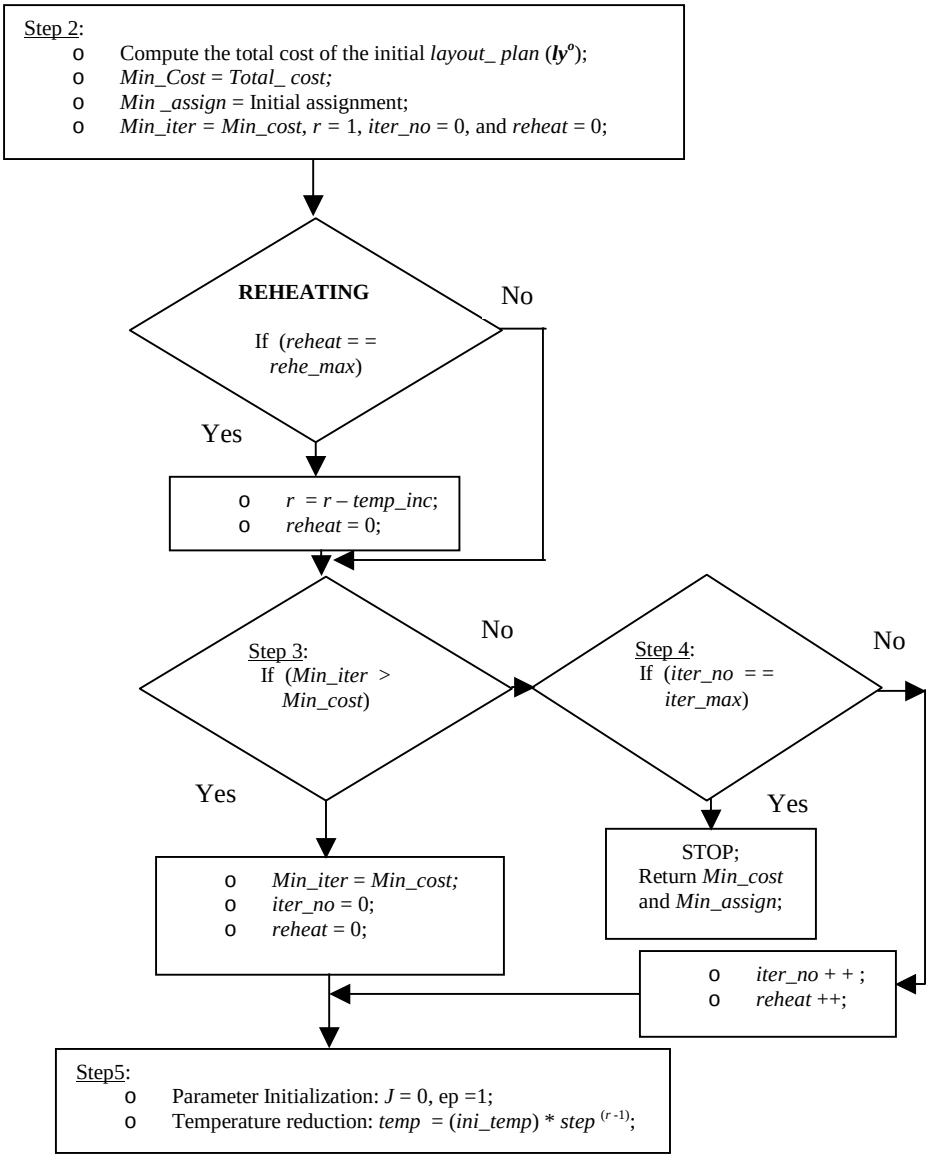


Figure 4.6: Reheating steps for the SA II algorithm

4.4 SA COMBO algorithm

The SA COMBO algorithm is a combination of heuristics such as the pairwise exchange heuristic with time windows (Urban, 1993), the SA algorithm, and the backward pass pairwise exchange heuristic (Balakrishnan and Cheng, 2000b). Urban (1993) pairwise exchange heuristic uses the forecasting window technique. In this heuristic, the flow data of the periods are combined and used to determine a layout arrangement for a particular period. Balakrishnan and Cheng (2000b) used a backward pass pairwise exchange heuristic to improve the solutions obtained by Urban (1993) heuristic. The general block diagram of the SA COMBO algorithm is given in figure 4.7.

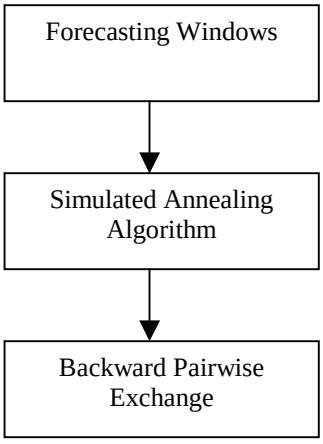


Figure 4.7: Block Diagram of SA COMBO

4.4.1 Concepts

4.4.1.1 Forecasting windows and the SA algorithm

The forecasting window size is the number of flow matrices combined prior to the implementation of the SA algorithm. The window size ranges from one to the number of periods, *T*. Given an initial layout, the SA algorithm improves the initial layout using the combined flow matrices.

An illustration of when the window size = 1 is given in figure 4.8. To improve an initial layout at period 1, the SA algorithm uses the flow data of the first period. Similarly, for the second period, the SA algorithm uses the flow data of the second period to improve the initial layout of period 2 (i.e., final layout of the first period). This procedure is continued until the final layout of the last period (i.e., fifth period) is obtained. For the fifth period, the SA algorithm uses the flow data of the fifth period to improve the initial layout of period 5 (i.e., final layout of the fourth period). Thus, a *layout_plan* is obtained when the window size = 1.

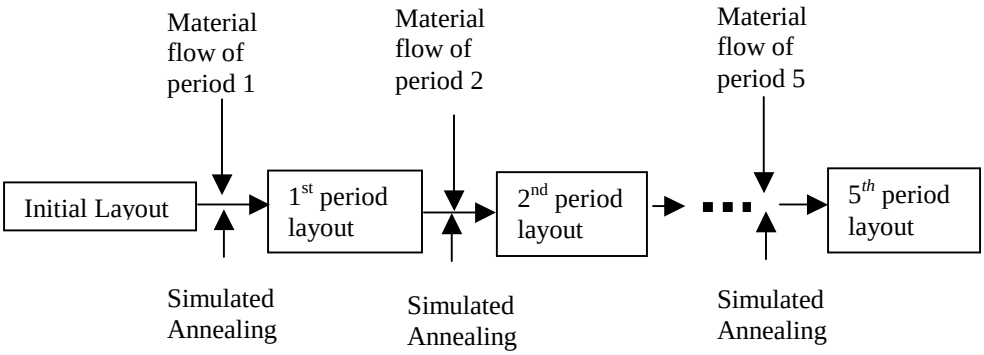


Figure 4.8: SA algorithm with forecast window size =1

Figure 4.9 illustrates the case when window size = 3. In period 1, the SA algorithm uses the combined flow matrices of periods 1, 2, and 3 to improve the given initial layout for period 1. Similarly, in period 2, the SA algorithm uses the combined flow matrices of periods 2, 3, and 4 to improve the initial layout for period 2 (i.e., final layout of the first period). Note, for the fourth period, the SA algorithm uses the combined flow matrices of periods 4 and 5 to improve the initial layout for period 4 (i.e., final layout of the third period). However, in period 5, the SA algorithm uses only the

flow data of the fifth period to improve the final layout of period 4 (i.e., initial layout of period 5). Therefore, for *T* windows (number of periods = *T*) there are *T layout_ plan*s, one for each window size. The *layout_plan* with the minimum cost is selected.

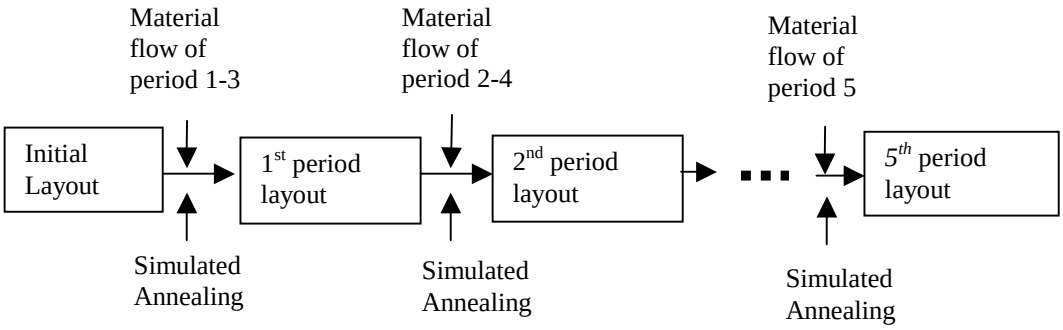


Figure 4.9: SA algorithm with forecast window size = 3

4.4.1.2 A modification of the SA Algorithm

The SA algorithm used with forecasting windows is similar to the SA algorithm for the SFLP (See section 2.2.3.3). In the SFLP, there are no rearrangements of departments. Therefore, the change in total cost is based only on the change in the flow cost. The computational formulas given in section 4.2.2.2.2 can be used to calculate the change in flow (Δ*flow*). However, in the SA algorithm with forecast windows, when two departments are exchanged, the calculation of the change in total cost is based on the change of the flow and rearrangement costs. An illustration is given in figure 4.10. The final layout of the first period of a 6-department problem is given where the window size = 1 and *T* = 3. The SA algorithm uses the flow matrix of period 2 to improve the final layout of the first period (i.e., initial layout of the second period). For instance, if departments 2 and 6 are selected randomly for exchange. The change in the flow cost can

be calculated. If these departments, departments 2 and 6, are exchange in the second period, these departments are going to be shifted (rearranged). Therefore, the rearrangement costs should be less than the improvement in the flow cost for period 2. In other words, $\Delta Total\_cost = \Delta flow$ – (rearrangement costs of departments 2 and 6) > 0.

2 3 4 6 1 5

Fig 4.10 Initial layout for the second period

4.4.1.3 Backward Pass Pairwise Exchange Heuristic

Balakrishnan and Cheng (2000b) proposed the backward pairwise exchange heuristic which may improve the results obtained by Urban's (1993) pairwise exchange heuristic with time windows. This heuristic produces "*m*" backward pass solutions (*layout_plan*s) based on "*m*" solutions (*layout_plan*s) produced by Urban's (1993) heuristic.

Similar to SA I and SA II, an initial layout plan (series of layouts) is given as input. The backward pass pairwise exchange starts at period *T*-1 and goes until the first period is reached. An illustration is given in figure 4.11. Figure 4.11 considers a 3-period ($T = 3$) *layout_plan* and explains the backward pass mechanism. The backward pass pairwise exchange heuristic starts with the second period (i.e., *T*-1). In the second period, the entire neighborhood is evaluated. In other words, for a 6-department problem, 15 neighborhood solutions are evaluated. The pair of departments selected for the exchange is the pair which gives the best improvement in the *Total_cost* (i.e., largest $\Delta Total\_cost$). Recall in section 4.2.2.3, $\Delta Total\_cost$ is based on the flow cost in the selected period and the rearrangement costs. The rearrangement costs from periods 2 and 3 (i.e., *T* and *T*+1),

and from periods 1 and 2 (i.e., *T*-1 and *T*) are considered. Thus, the backward pass mechanism considers two periods (previous and future) while evaluating the current period. Hence, each of the *m* solutions is at least as good as Urban's (1993) heuristic solutions.

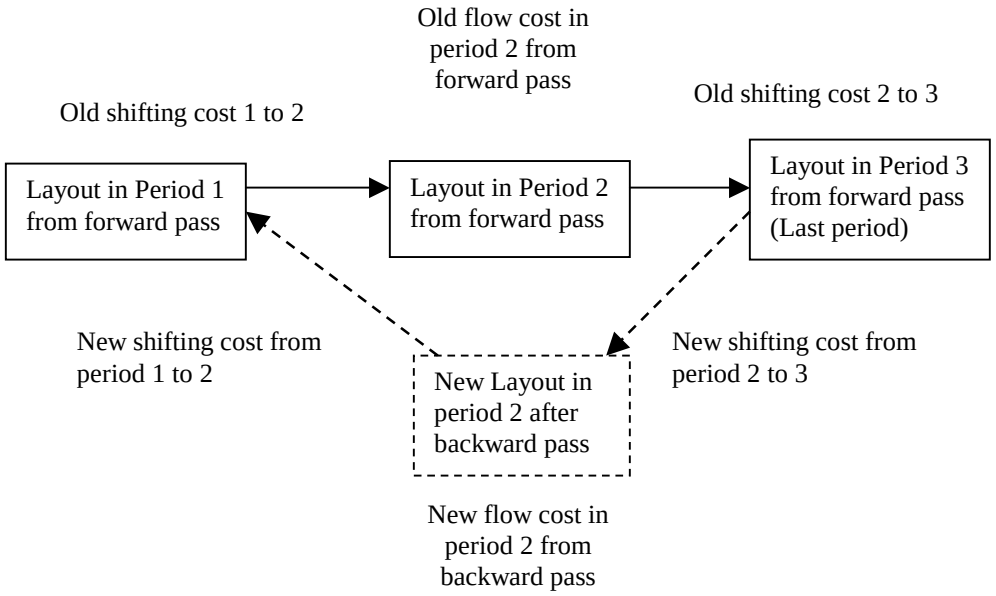


Figure 4.11: Backward Exchange Mechanism

The Backward pass differs from Urban's (1993) heuristic in another aspect. In figure 4.12, a block diagram of both Urban's (1993) heuristic and backward pass pairwise exchange heuristic of (Balakrishnan and Cheng, 2000b) for a 3-period ($T = 3$) problem is given. In Urban's heuristic, the material flow is added (combined) according to the window size. However, in the backward pass, only the material flow associated with the respective period is used. In other words, the window size is always equal to one in the backward pass method.

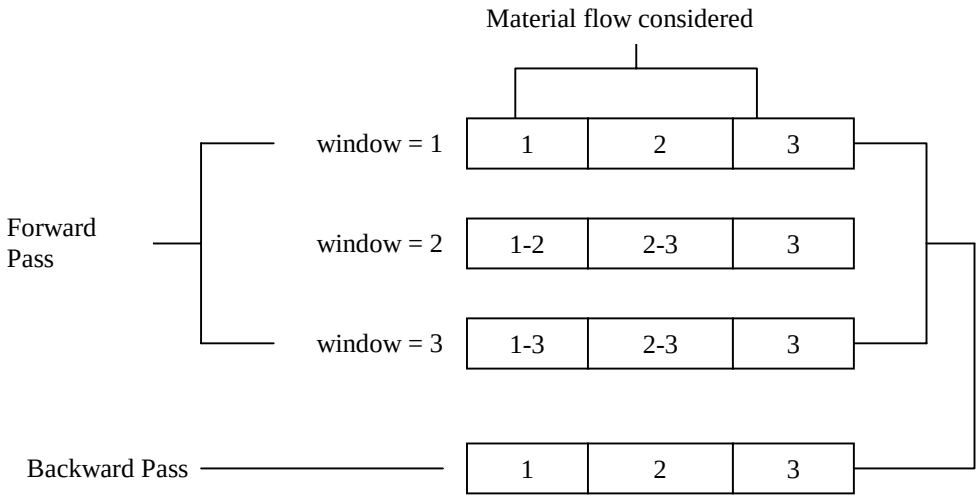


Figure 4.12: Backward pass with window size = 3

4.4.1.4 An illustration of the backward pass mechanism

Consider a problem instance where $N = 6$ and $T = 3$. Since $T = 3$, there are three *layout_ plan*s provided by the forward pass algorithm (i.e., by the SA algorithm with forecast windows). Figure 4.13 represents one of the layout plans. The backward pass pairwise exchange heuristic starts in the second period. For instance, the exchange of departments 1 and 2 ($u = 1$ and $v = 2$) improves the flow cost in the second period. Moreover, this exchange reduces the total rearrangement costs. Hence, *Total_cost* is reduced. Therefore, the exchange is made. As mentioned in the previous section, a swap in the second period is related to the flow cost of the second period and the rearrangement costs from period 1 to 2 and from period 2 to 3. The total cost of a *layout_plan* consists of the flow cost of all the periods and the total rearrangement cost incurred from period to period. Thus, the swap that improves the flow cost of any period actually improves the total flow cost but it may or may not decrease the total rearrangement cost.

2 3 4 6 1 5 - Period 1
1 2 3 4 5 6 - Period 2
4 5 6 1 3 2 - Period 3

Figure 4.13: Before the exchange

As previously mentioned, since the exchange of departments 1 and 2 in period 2 reduces the *Total_cost*, the exchange is made. See figure 4.14, which represents the *layout_plan* after the exchange. The current *layout_plan* shows that the total rearrangement cost is reduced, since department 1 is assigned to location 2 during periods 1 and 2.

2 3 4 6 1 5 - Period 1
2 1 3 4 5 6 - Period 2
4 5 6 1 3 2 - Period 3

Figure 4.14: After the exchange (current *layout_ plan*)

4.4.2 SA COMBO Nomenclature

The parameters of the SA algorithm in SA COMBO is the same as the SA I algorithm discussed in section 4.2.3. The following are additional notation for the SA COMBO algorithm.

window = number of windows = T = number of periods;

w_size = counter used to count the window size. Maximum limit on this counter is set equal to *window*;

p_ size = counter used to count the number of the periods. Maximum limit on this counter is set equal to T;

bwp_size = counter which represents the period considered in the backward pairwise exchange heuristic;

4.4.3 Algorithm Exposition

See figure 4.15 for the SA COMBO algorithm, and the steps for the algorithm are as follows:

Step 1:

The algorithm starts with setting *w_size* = *p_size* = 1. An initial layout (i.e., initial layout of the first period) is given. The SA algorithm improves the initial layout to obtain the final layout of the first period.

Step 2:

Flow matrices are added according to *p_size* and *w_size*. Since *w_size* = *p_size* = 1, only one matrix is considered (i.e., flow matrix of the first period).

Step 3:

The SA algorithm (SA I) is applied to the initial layout. The parameters: epoch (*e*), initial temperature (*ini_temp*), number of non-improvement iterations (*iter_max*), step length (*step*), and the maximum limit on *J* are set at the correct levels (i.e., parameters must be fine-tuned). As mentioned in section 4.4.1.2, this SA algorithm is modified to calculate the change in the total cost.

Step 4:

The final layout is added to the *layout_ plan* with respect to the *p_size*. For instance, when *p_size* = 1, *layout _ plan* = [*ax*(1)] (i.e., only the final layout of the first period is obtained). This layout is used as an initial layout for the second period (i.e., when *p_size* = 2).

Step 5:

The counter *p_size* is compared with the number of time periods (*T*). If *p_size* < *T*, then the *p_size* is incremented by one. Then, go to step 2. If the condition is not satisfied, then go to step 6 (i.e., the complete *layout_plan* is obtained and it is given as input for the backward pass pairwise exchange heuristic).

Step 6:

The *layout_plan* constructed is used as input for the backward pass pairwise exchange algorithm. This algorithm starts at period $T - 1$. The period is decremented (backward) by one. The counter *bwp_size* is used to count the period inside the backward pass algorithm, and it is set equal to $T - 1$ (i.e., $bwp\_size = T - 1$).

Step 7:

Note, only the flow matrix corresponding to *bwp_size* is considered. In other words, the time windows concept is not used in the backward pass pairwise exchange heuristic.

Step 8:

The neighborhood of the current solution (layout at *bwp_size* period) is evaluated. All possible pairs of departments are evaluated, and the pair which results in the largest *Total_cost* reduction is selected. The $\Delta Total\_cost$ (i.e., the rearrangement costs with respect to the exchange is deducted from the $\Delta flow$) is obtained.

Step 9:

If $\Delta Total\_cost > 0$, then the exchange is implemented. Go to step 8. Otherwise, go to step 10.

Step 10:

The *bwp_size* is checked to verify the conclusion of the backward pass algorithm (i.e., when *bwp_size* reaches 1). If *bwp_size* = 1, go to step 11. Otherwise, *bwp_size* is decremented by one.

Step 11:

If *w_size* $\leq$ *window*, then the *p_size* is initialized to one, and the *w_size* is incremented. Go to step 2. The entire process is repeated for the new *w_size*. Otherwise, all the window sizes have been explored; therefore, go to step 12.

Step 12:

For each *w_size*, a *layout_plan* is obtained. Therefore, for *T* windows, *T* *layout_plan*s are obtained. The *layout_plan* with the minimum cost is selected and the algorithm terminates.

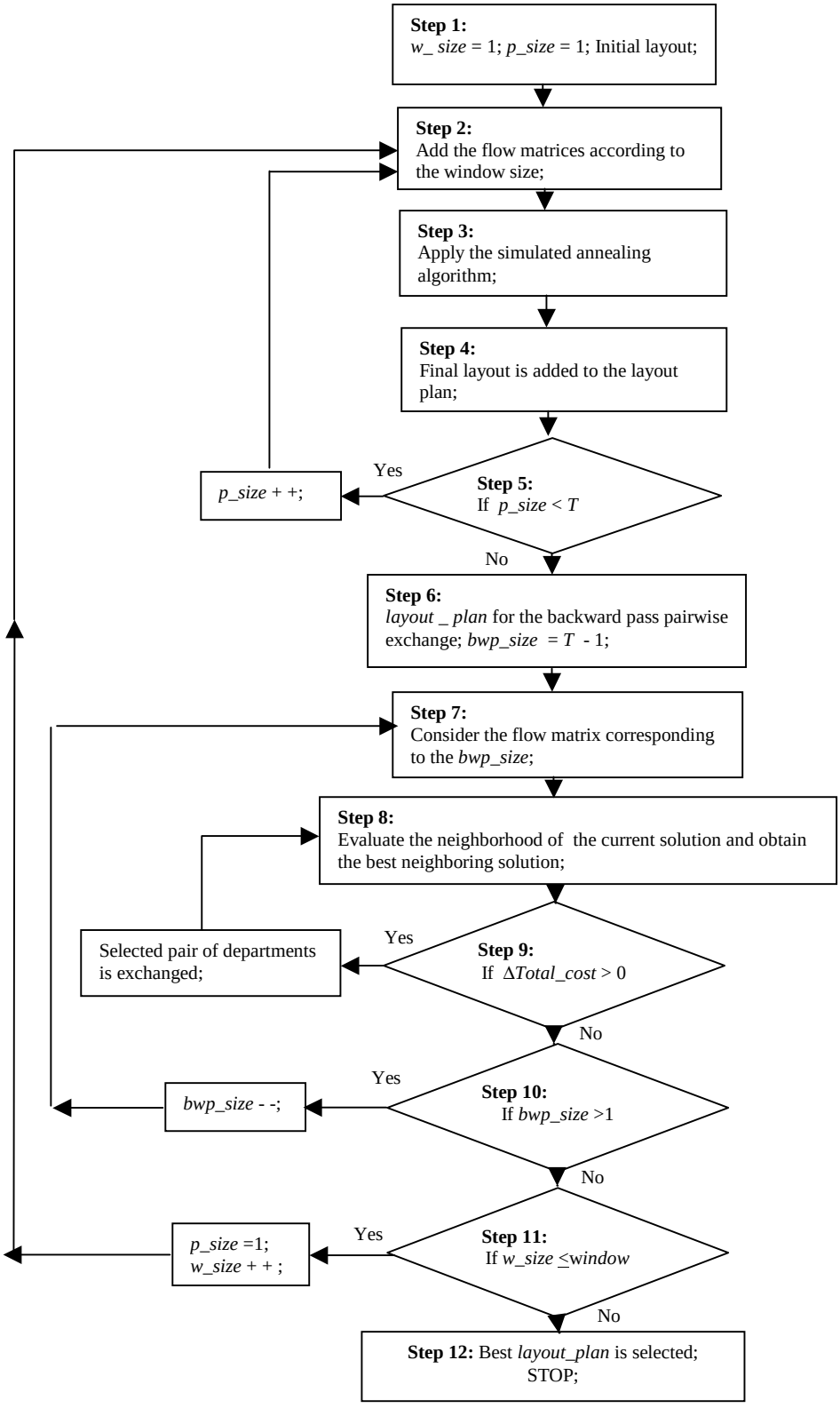


Figure 4.15: SA COMBO algorithm

4.4.4 The Results obtained from the SA COMBO algorithm

The output obtained by implementing the SA COMBO algorithm on the problem instance used in section 4.2.5 is given in figure 4.16. The problem instance has 9 departments (N = 9) and 5 time periods (T = 5). The parameters of the SA algorithm such as *NN*, *NS*, *iter_max*, *step*, and *ini_temp* are set at levels 10, 500, 1000, 0.95, 200,000, respectively. Also, an initial assignment is given for the first period. Note the levels of the above parameters are not obtained from experimental evaluation. The levels are for the purpose of illustration.

The solution of the SA COMBO algorithm for the problem instance has five different *layout_plan*s. As mentioned previously, the SA algorithm with time windows produces a *layout_plan* for each window size, and the *layout_plan* is improved by using the backward pass pairwise exchange heuristic. For instance, when *w_size* = 2, the backward pass pairwise exchange heuristic starts at period 4. The backward pass pairwise exchange heuristic evaluates the entire neighborhood of solutions (36 solutions) for exchange. Since none of the pair of departments improves the total cost, the layout remains unchanged. However, at period 1, the pair of departments 3 and 9 is selected for exchange, since it improves the total cost of the *layout_plan*. See figure 4.16. Similarly, when the window size = 3, the backward pass heuristic improves the total cost of the *layout_plan* by improving the layout of the first period. In conclusion, four *layout_plan*s (i.e., *w_size* = 2, 3, 4, and 5) obtained by SA with time windows are improved by the backward pass algorithm except the *layout_plan* obtained when *w_size* = 1.

In the next chapter, the proposed algorithms are tested on a set of test problems taken from the literature. Also, the results are discussed.

Initial assignment for the first period
1 2 3 4 5 6 7 8 9

layout_plan by SA with time window = 1
3 5 6 9 2 1 4 7 8
8 6 5 3 2 1 9 7 4
1 6 7 5 8 9 2 3 4
8 4 5 6 2 1 3 9 7
9 4 5 3 6 2 7 8 1
Totalcost = 613229
layout_plan after the backward pass heuristic
3 5 6 9 2 1 4 7 8
8 6 5 3 2 1 9 7 4
1 6 7 5 8 9 2 3 4
8 4 5 6 2 1 3 9 7
9 4 5 3 6 2 7 8 1
Totalcost = 613229

layout_plan by SA with time window = 2
3 5 2 8 6 9 7 1 4
8 1 5 4 6 3 7 9 2
7 6 8 5 2 3 1 4 9
9 2 7 8 4 5 1 6 3
1 8 5 3 2 6 7 4 9
Totalcost = 623741
layout_plan after the backward pass heuristic
3 5 4 8 6 9 7 1 2
8 1 5 4 6 3 7 9 2
7 6 8 5 2 3 1 4 9
9 2 7 8 4 5 1 6 3
1 8 5 3 2 6 7 4 9
Totalcost = 621666

layout_plan by SA with time window = 3
1 5 4 2 8 9 3 6 7
9 4 8 5 2 1 3 6 7
9 4 8 5 2 1 3 6 7
3 4 9 6 8 5 7 2 1
3 4 8 6 9 5 7 2 1
Totalcost = 631452

Continue *layout_plan* by SA with time window = 3
layout_plan after the backward pass heuristic
6 5 4 3 8 9 7 1 2
9 4 8 2 5 1 3 6 7
9 4 8 5 2 1 3 6 7
3 4 9 6 8 5 7 2 1
3 4 8 6 9 5 7 2 1
Totalcost = 621662

layout_plan by SA with time window = 4
7 5 4 8 6 3 9 2 1
3 4 2 5 8 7 9 6 1
3 4 2 5 8 7 9 6 1
1 8 3 2 6 5 9 4 7
3 4 5 9 6 8 1 2 7
Totalcost = 630086
layout_plan after the backward pass heuristic
7 5 2 8 6 3 9 1 4
3 4 2 8 5 7 9 6 1
3 4 2 5 8 7 9 6 1
1 8 3 2 6 5 9 4 7
3 4 5 9 6 8 1 2 7
Totalcost = 621327

layout_plan by SA with time window = 5
1 5 2 6 8 9 7 4 3
3 8 6 5 4 7 1 2 9
3 8 6 5 4 7 1 2 9
7 2 9 8 6 5 3 4 1
3 4 5 9 6 8 1 2 7
Totalcost = 634856
layout_plan after the backward pass heuristic
3 5 4 6 8 9 7 1 2
3 8 6 4 5 7 1 2 9
3 8 6 5 4 7 1 2 9
7 2 9 8 6 5 3 4 1
3 4 5 9 6 8 1 2 7
Totalcost = 622711

Figure 4.16: Output from the SA COMBO algorithm

CHAPTER 5

COMPUTATIONAL EXPERIMENTS AND RESULTS

5.1 Data Set

The computational experiments were conducted on the data set provided by Lacksonen and Enscore (1993). The data set was constructed based on several factors: percentage of new departments/period, number of departments, number of time periods, ratio of rearrangement cost and flow cost (rearrangement/flow cost), percentage of positive flow, and maximum value of flow changes per period. In this data set, the rearrangement cost is the same for all departments and is constant across all periods. Additionally, when new departments replaced old departments, the replacement costs are considered, and the cost of replacing a department is equal to the rearrangement cost.

The data set includes DFLP instances involving 6, 12, 20, and 30 departments (i.e., N = 6, 12, 20, and 30), with 3 and 5 periods (i.e., T = 3 and 5). Each problem with department size (N) has four 3-period problem instances and four 5-period problem instances (Lacksonen and Enscore, 1993). Thus, 32 problem instances are available in this data set. These test problems are labeled from P01 to P32. Sample input data, for problem instance P01, are given in appendix B.

5.2 Algorithms

The proposed heuristics (i.e., the SA algorithm, SA I; the SA algorithm with reheating, SA II; and the SA algorithm combined with other heuristics, SA COMBO) were tested on the set of test problems. All these algorithms were coded in C and solved on a Pentium 300Mhz PC. The C code for the algorithms proposed in this research (SA I, SA II, and SA COMBO) are given in appendix C. The initial solutions (layouts) for all

these algorithms are as follows: the initial layout for the first period is given as $ax(1) = (1, 2, 3, 4, 5,\ldots, N)$, and the same layout is given for the rest of the periods. Therefore, the total replacement cost (*Rep_cost*) of the *layout_plan* is zero. For the SA COMBO algorithm, the same layout (i.e., $ax(0) = (1, 2, 3, 4, 5,\ldots, N)$) is given as an initial layout for the first period. The improved solution for the first period layout is used as the initial solution for the second period layout and so on. This process continues until the final layout of the last period is obtained.

5.3 Parameter Settings

The parameters such as epoch length, maximum number of non-improvement iterations, and step length (cooling rate) were kept at the same level for both the SA I and the SA II algorithms. However, the SA II algorithm differs from SA I since it has additional parameters such as *rehe_max* and *temp_inc*. The SA COMBO heuristic was tested using two different settings: the number of non-improvement iterations, epoch length, and step length were set at the lower levels (SA COMBOa) and these parameters were set at the higher levels (SA COMBOb). See table 5.1 for the parameter settings. The parameter levels were determined based on an extensive experimental evaluation of the proposed heuristics. Note, for the 30-department problems the SA II algorithm used the following settings: *rehe_max* = 300 and *temp_inc* = 100, and for the other problems, *rehe_max* = 100 and *temp_inc* = 50.

Table 5.1 Parameter Settings

Parameters	**SA I**	**SA II**	**SA COMBOa**	**SA COMBOb**
ini_temp	*initial_cost*	*initial_cost*	*initial_cost*	*initial_cost*
Step	0.99	0.99	0.95	0.99
NN	10	10	10	10
NS	900	900	500	900
iter_max	1900	1900	1000	1900
rehe_max	-	100 \ 300	-	-
temp_inc	-	50 \ 100	-	-

5.4 Experimental results

Each setting of the proposed heuristics was implemented three times for each problem instance and the best out of the three trials were reported. Table 5.2 summarizes the results obtained from solving the set of test problems using the proposed heuristics. The results obtained by the TS heuristic (Kaku and Mazzola, 1997), cutting planes (CP) algorithm (Lacksonen and Enscore, 1993), and UB (Kaku and Mazzola's (1997) version of Urban's (1993) heuristic) are also given in Table 5.2. The shaded areas, before the 'SA Best' column, in Table 5.2 highlight the 'SA best' solutions found by one of the algorithms (SA I, SA II or SA COMBO). If all of the algorithms obtained the 'SA Best' solution, then no area is shaded for that problem instance. The shaded areas, after the 'SA Best' column, highlight the best-found solutions obtained by one of the algorithms (CP or TS). If both the algorithms obtained the best-found solution, then no area is shaded for that problem instance. For the P31 problem instance, the solutions obtained by the SA I, SA II, SA COMBOa and SA COMBOb are: 12404, 12461, 12148, and 12148, respectively. Since SA COMBOa and SA COMBOb give the best solution ('SA Best'), both of the solutions are shaded. Also, for the same problem instance, CP and TS

obtained the following solutions: 12351 and 12163, respectively. Since the solution obtained by TS is better than the CP solution, the TS solution is shaded. Note for this problem instance, one of the proposed heuristics obtained the best-found solution ever (i.e., SA best is lesser than the Best-found).

The best-found solutions are largely obtained using the TS and the CP algorithms. Although the CP method performed as well as or better than TS for problem instances with 6, 12, and 20 departments, TS clearly outperformed the CP method for problem instances with 30 departments. For most of the problem instances, the SA COMBO algorithm performed better than the SA I and SA II algorithms. More specifically, for the 20 department problem instances, the SA COMBO obtained better results than the SA I and SA II algorithms. The best solutions obtained by the proposed heuristics (i.e., SA I, SA II, and SA COMBO) were compared to the best-found solutions available in the literature. The proposed heuristics obtained the best-found solution for 17 problem instances and found the best solution ever for one of the problem instances P31. Also, in problem instance P27, SA COMBO outperformed TS, but did not obtain a better solution than the CP algorithm. For 8 problem instances, the solutions obtained by the proposed heuristics were within 1% of the best-found solutions. Also, for 6 problem instances, the solutions obtained were within 1-2% of the best-found solutions.

In problem instances P06, P14, P22, and P30, the SA I and SA II algorithms gave better results than the SA COMBO algorithm. Note these problems are considered to be the most complex problem instances in the data set because of the high number of time periods, high percentage of new departments, high percentage of positive flows,

maximum value of flow changes, and low ratio of rearrangement cost and flow cost (i.e., require most rearrangements).

The proposed heuristics outperformed the TS heuristic in two problem instances (P27 and P31) and performed equally as well for 57% of the test problems. Furthermore, the proposed heuristics performed as well as or better than TS for 59% of the test problems. Moreover, the computational times required for the proposed heuristics is much less than the TS heuristic (See Table 5.3). The computational times given in table 5.3 are the average time required to solve the 3 and 5 period test problems. Note, the TS (Kaku and Mazzola, 1997) heuristic were tested on a Pentium 200MHz PC and the proposed heuristics were tested on Pentium 300 MHz PC. In the next chapter, the research concludes with the areas for future research.

Table 5.2 Results from the proposed, UB, CP, and TS heuristics

Problem Size											Best	% within best-
Size	Time period	P No	SA I	SA II	SA COMBOa	SA COMBOb	SA Best	UB	CP	TS	Found	found solution
6	3	P01	267	267	267	267	267	291	267	267	267	0
		P02	260	260	260	260	260	281	260	260	260	0
		P03	363	363	363	363	363	385	363	363	363	0
		P04	299	299	299	299	299	312	299	299	299	0
		P05	442	442	442	442	442	472	442	442	442	0
	5	P06	**596**	**596**	597	597	596	617	589	**586**	586	-1.706
		P07	424	424	424	424	424	500	424	424	424	0
		P08	428	428	428	428	428	452	428	428	428	0
12	3	P09	1624	1624	1624	1624	1624	1676	1624	1624	1624	0
		P10	1980	**1973**	**1973**	**1973**	1973	2000	1973	1973	1973	0
		P11	1661	1661	1661	1661	1661	1779	1661	1661	1661	0
		P12	2106	**2105**	**2105**	**2105**	2105	2298	2097	2097	2097	-0.381
		P13	**2966**	**2966**	2977	2977	2966	3050	2930	2930	2930	-1.229
	5	P14	**3715**	**3715**	3738	3748	3715	3886	3726	**3701**	3701	-0.378
		P15	2756	2756	2756	2756	2756	3125	2756	2756	2756	0
		P16	3382	**3364**	**3364**	**3364**	3364	3730	3364	3364	3364	0
20	3	P17	2764	2761	2761	**2758**	2758	2925	2763	**2758**	2758	0
		P18	5342	5352	**5318**	**5318**	5318	5363	5318	5318	5318	0
		P19	3151	3102	**3064**	**3064**	3064	3924	**3048**	3056	3048	-0.525
		P20	6036	6042	**5947**	**5947**	5947	7147	**5873**	5903	5873	-1.26
		P21	4749	4702	**4663**	4676	4663	4964	**4581**	4605	4581	-1.76
	5	P22	9963	**9897**	9929	9929	9897	10530	9825	**9746**	9746	-1.549
		P23	4945	4738	**4654**	**4654**	4654	6512	4654	4654	4654	0
		P24	9188	9075	**8979**	**8979**	8979	10816	8985	**8979**	8979	0
30	3	P25	7185	**7131**	**7131**	**7131**	7131	7516	7163	**7130**	7130	-0.014
		P26	**14519**	14538	15123	15142	14519	14729	14583	**14478**	14478	-0.283
		P27	8184	8185	8122	**8093**	8093	10465	**8066**	8115	8066	-0.335
		P28	15219	15192	15095	**15074**	15074	15285	14940	**14925**	14925	-0.998
		P29	**13753**	13761	13772	13760	13753	14103	13719	**13606**	13606	-1.08
	5	P30	25754	**25721**	26094	26084	25721	26223	26027	**25583**	25583	-0.539
		P31	12404	12461	**12148**	**12148**	12148	15738	12351	**12163**	12163	0.123
		P32	24402	24498	**24200**	**24200**	24200	27680	24409	**24200**	24200	0

Table 5.3 Computaional time of proposed and TS heuristics (time is given minutes)

Problem Size							
Size	**Time period**	**P No**	**SA I**	**SA II**	**SA COMBOa**	**SA COMBOb**	**TS**
6	3	P01	0.814	0.7438	1.3646	3.6684	0.0133
		P02					
		P03					
		P04					
	5	P05	0.8831	0.847	3.775	10.4763	0.0133
		P06					
		P07					
		P08					
12	3	P09	2.6275	5.4683	3.143	8.9147	1.7
		P10					
		P11					
		P12					
	5	P13	2.8613	2.909	9.4816	24.0217	3.8167
		P14					
		P15					
		P16					
20	3	P17	7.9152	8.0109	5.9763	17.4302	16.233
		P18					
		P19					
		P20					
	5	P21	8.94	8.545	16.4471	46.9112	35.7667
		P22					
		P23					
		P24					
30	3	P25	20.8897	20.7302	10.3933	30.4552	70.7167
		P26					
		P27					
		P28					
	5	P29	20.3738	23.5472	28.3577	80.3775	167.15
		P30					
		P31					
		P32					

CHAPTER 6

CONCLUSIONS

6.1 Introduction

The objective of this research was to develop heuristics to obtain good solutions for the DFLP. Three heuristics (SA I, SA II and SA COMBO) were used to solve the DFLP. The performance of these algorithms was compared with recent heuristics such as the TS heuristic and the pairwise exchange heuristics with time windows. The next section summarizes the research, and the subsequent sections conclude the research with its contributions and recommendations for future research.

6.2 Summary of the Research

The DFLP is defined, and the assumptions of the problem are given. In the DFLP, the layout plan is a series of layouts for a specified time horizon. During this time horizon, if material flow changes several times (e.g., 5 times), then the facility may be rearranged for each period. The analysis is based on the tradeoffs between the costs of excess material handling (occurs if a layout is not rearranged when it is required) and the costs of such rearrangements. Thus, the objective is to minimize the sum of the flow costs and rearrangement costs throughout the planning horizon.

Three heuristics (SA I, SA II, and SA COMBO) are presented in details. The SA I algorithm is a straightforward implementation of SA, and the SA II algorithm is the same as the SA I algorithm but with reheating. Given an initial solution (layout plan), SA I and SA II algorithms improve the initial solution to obtain better solutions. The SA COMBO algorithm is the combination of the pairwise exchange heuristic with time windows, SA, and the backward pass pairwise exchange heuristics. The time windows technique is used

to combine (sum) the flow matrices for 1 or more periods. Then the SA heuristic is implemented using the combined flow matrix. For each length of the forecast window (m = 1, . . . , T where T = number of periods), the SA algorithm obtains a layout plan, and this layout plan is improved by the backward pass pairwise exchange heuristic.

The proposed heuristics were tested on a data set of 32 problem instances taken from the literature, and the results are presented. The performance of the proposed heuristics is evaluated using two measures: solution quality and time. The results show that the proposed heuristics are efficient. The proposed heuristics obtained the best-found solutions for 17 problem instances and obtained the best solution ever for one problem instance. Furthermore, in 8 problem instances, solutions obtained are within 1% of the best-found solutions, and for 6 problem instances, solutions obtained are within 1-1.8% of the best-found solutions. Also, the proposed heuristics obtained the solutions in reasonable time.

6.3 Contribution of the Research

The SA COMBO algorithm is a combination of the pairwise exchange heuristic with time windows technique, SA algorithm, and the backward pass pairwise exchange heuristic. Computational experience showed that this is an efficient heuristic for solving the DFLP. Moreover, the comparison of the proposed heuristics with other heuristics such as the pairwise exchange heuristic with time windows, TS heuristic, and the cutting plane algorithm shows that the heuristics developed perform well.

6.4 Recommendations for Future Research

The following recommendations are given for future research:

- Rearrangement costs that depend on time periods (time-value of money), locations (distance), and number of machines moved can be considered.
- The development of hybrid techniques for the DFLP can be considered. The combination of key elements of the TS heuristic and SA algorithm may produce better results. Furthermore, the elements of the genetic algorithm can be combined with the key elements of the SA algorithm or TS heuristic.
- Neural network techniques and recent heuristics such as the Ant colony system have not yet been applied to the DFLP and can be considered.
- The dynamic layout problem where the flow between departments is stochastic is rarely considered. This is another area for future research.

REFERENCES

Ahuja, R.K., Orlin, J.B., and Tiwari, A., 2000, "A greedy genetic algorithm for the Quadratic Assignment problem, " *Computers and Operations Research*, Vol. 27, pp. 917-934.

Armour, G.C. and Buffa, E.S., 1963, "A heuristic algorithm and simulation approach to relative location of facilities," *Management Science*, Vol. 39, No. 1, pp. 294-309.

Balakrishnan, J. and Cheng, C.H., 1998, "Dynamic Layout Algorithms: a State-of-the art Survey", *Omega*, Vol. 26, No. 4, pp. 507-521.

Balakrishnan, J. and Cheng, C.H., 2000a, "Genetic search and the dynamic layout problem," *Computers and Operations Research*, Vol. 27, pp. 587-593.

Balakrishnan, J. and Cheng, C.H., 2000b, "An improved pairwise exchange for the dynamic plant layout problem," *International Journal of Production Research*, Vol. 38, No. 13, pp. 3067-3077.

Balakrishnan, J., Jacobs, F.R., and Venkataramanan, M.A., 1992, "Solutions for the constrained dynamic facility layout problem," *European Journal of Operational Research*, Vol. 57, pp. 280-286.

Ballou, Ronald, 1968, "Dynamic Warehouse Location Analysis," *Journal of Marketing Research*, Vol. 5, No. 3, pp. 271-276.

Banerjee, P., Montreuil, B., Moodie, C.L., and Kashyap, R.L., 1992, "A Modeling of Interactive Facilities Layout Designer Reasoning Using Qualitative Patterns," *International Journal of Production Research*, Vol. 30, No. 3, pp. 433-453.

Battiti, R. and Tecchioli, G., 1994, "Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks," *Computers and Mathematics with applications*, Vol. 28, No. 6, pp. 1-8.

Baykosaglu, A. and Gindy, N.N.Z., 2001, "A simulated annealing algorithm for dynamic facility layout problem," *Computers and Operations Research*, Vol. 28, pp. 1403-1426.

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

Bazarra, M.S. and Sherali, M.D., 1980, "Bender's partitioning scheme applied to a new formulation of the quadratic assignment problem," *Naval Research Logistics Quarterly*, Vol. 27, No. 1, pp. 29-41.

Bean, J.C., 1994, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, Vol. 6, pp. 154-160.

Block, T.E., 1978, "Fate: A new construction algorithm for facilities layout," *Journal of Engineering Production*, Vol. 2, pp. 111-120.

Bozer, Y.A. and Meller, R.D., 1994, "An Improvement Type Layout Algorithm for Single and Multiple Floor Facilities," *Management Science*, Vol. 40, No. 7, pp. 918-932.

Burkard, R.E. and Bonniger, T., 1983, "A heuristic for quadratic boolean program with application to Quadratic Assignment problems," *European Journal of Operational Research*, Vol. 13, pp. 374-386.

Burkard, R.E. and Rendl, F., 1984, "A Thermodynamically motivated Simulation Procedure for Combinatorial Optimization Problems," *European Journal of Operational Research*, Vol. 17, pp. 169-174.

Burkard, R.E. and Stratman, K.H., 1978, "Numerical investigations on Quadratic Assignment Problems," *Naval Research Logistics Quarterly*, Vol. 25, pp. 129-144.

Carrie, A.S., Moore, J.M., Roczniak, M., and Seppanen, J.J., 1978, "Graph Theory and computer-aided facilities design," *Omega*, Vol. 6, No. 4, pp. 353-361.

Chiang, W.C. and Chiang, C., 1998, "Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation," *European Journal of Operational Research*, Vol. 106, pp. 457-488.

Chiang, W.C. and Kouvelis, P., 1996, "An improved tabu search heuristic for solving facility layout design problems," *International Journal of Production Research*, Vol. 34, No. 9, pp. 2565-2585.

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

Conway, D.G. and Venkataramanan, M.A., 1994, "Genetic search and the dynamic layout problem," *Computers and Operations Research*, Vol. 21, No. 8, pp. 955-960.

Deisenroth, M.P. and Apple, J.M., 1972, "A computerized plant layout analysis and evaluation technique," Technical Paper, Annual AIIE Conference, Norcross, GA.

Dowsland, K.A., 1993, "Some experiments with simulated annealing techniques for packing problems," *European Journal of Operational Research*, Vol. 68, pp. 389-399.

Drezner, Z., 1980, "Discon: A new method for the layout problem," *Operations Research*, Vol. 25, No. 6, pp. 1375-1384.

Eades, P., Foulds, L.R., and Giffin, J., 1982, "An efficient heuristic for identifying a maximum weight planar subgraph," in: Lecture Notes in Mathematics No. 952 (Combinatorial Mathematics IX), Springer, Berlin.

Edwards, H.K., Gillet, B.E., and Hale, M.C., 1970, "Modular Allocation technique (MAT)," *Management Science*, Vol.17, No. 3, pp. 161-169.

Elshafei, A.N., 1977, "Hospital Layout as a quadratic assignment problem," *Operations Research Quarterly*, Vol. 28, No. 1, pp. 167-179.

Fleurent, C. and Ferland, J.A., 1994, "Genetic Hybrids for the Quadratic Assignment Problem," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 16, pp. 173-188.

Fortenberry, J.C. and Cox, J.F., 1985, "Multiple Criteria Approach to the Facility Layout Problem," *International Journal of Production Research*, Vol. 23, pp. 773-782.

Foulds, L.R. and Robinson, D.F., 1976, "A strategy for solving the plant layout problem," *Operations Research*, Vol. 27, No. 4, pp. 845-855.

Foulds, L.R. and Robinson, D.F., 1978, "Graph Theoretic heuristics for the plant layout problem," *International Journal of Production Research*, Vol. 16, No.1, pp. 27-37.

Francis, R.L. and White, J.A., 1974,"*Facility Layout and Location: An Analytical Approach*," Englewood Cliffs, NJ: Prentice-Hall.

Francis, R.L., McGinnis Jr., L.F., and White, J.A., 1992,"*Facility Layout and Location: An Analytical Approach*," Englewood Cliffs, NJ: Prentice-Hall.

Gilmore, P.C., 1962, "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *Journal of the Society for the Industrial and Applied Mathematics*, Vol. 10, pp. 305-313.

Glover, F., 1989, "Tabu Search- Part I," *ORSA Journal on Computing*, Vol. 1, No. 3, pp. 190-206.

Glover, F., 1990, "Tabu Search- Part II," *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 4-32.

Goetschalckx, M., 1992, "SPIRAL: An efficient and Interactive Adjacency Graph Heuristic for Rapid Prototyping of Facilities Design," *European Journal Of Operational Research*, Vol. 63, No. 2, pp. 304-321.

Hassan, M.M.D. and Hogg, G.L., 1987, "A Review of Graph Theory Applications to the Facilities Layout Problem," *Omega*, Vol. 14, No. 4, pp. 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*, Vol. 27, No.7, pp. 1149-1160.

Hassan, M.M.D., Hogg, G.L., and Smith, D.R., 1986, "SHAPE: A Construction Algorithm for Area Placement Evaluation," *International Journal of Production Research*, Vol. 24, pp. 1283-1295.

Heragu, S.S., 1992, "Recent models and techniques for solving the layout problem," *European Journal of Operational Research*, Vol. 57, No. 2, pp. 136-144.

Heragu, S.S. and Alfa, A.S., 1992, "Experimental analysis of simulated annealing based algorithms for the layout problem," *European Journal of Operational Research*, Vol. 57, pp. 190-202.

Heragu, S.S. and Kusiak, A., 1991, "Efficient Models for the facility layout problem," *European Journal of Operational Research*, Vol. 53, No. 1, pp. 1-13.

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

Hillier, F.S. and Connors, M.M., 1966, "Quadratic Assignment problem algorithms and the location of indivisible facilities," *Management Science*, Vol. 13, pp. 42-57.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C., 1989, "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Operations Research*, Vol. 37, No. 6, pp. 865-892.

Kaku, B.K. and Mazzola, J.B., 1997, "A Tabu search Heuristic for the Dynamic Plant Layout Problem," *Informs Journal on Computing*, Vol. 9, No.4, pp. 374-384.

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

Kelly, J.P., Laguna, M., and Glover, F., 1994, "A study of diversification strategies for the Quadratic Assignment Problem," *Computers and Operations Research*, Vol. 21, No. 8, pp. 885-893.

Khalil, T.M., 1973, "Facilities relative allocation technique (FRAT)," *International Journal Production Research*, Vol. 11, No. 2, pp. 183-194.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, 1983, "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-680.

Koopmans, T.C. and Beckman, M.J., 1957, "Assignment Problems and the Location of economic Activities," *Econometrics*, Vol. 25, No. 1, pp. 53-76.

Kouvelils, P., Kurawarwala, A.A., and Gutierrez, G.J., 1992, "Algorithms for Single and Multiple Period Layout Planning for Manufacturing Systems," *European Journal of Operational Research*, Vol. 63, pp. 287-303.

Kusiak, A. and Heragu, S.S., 1987, "The facility layout problem," *European Journal of Operational Research*, Vol. 29, pp. 229-251.

Lacksonen, T.A. and Enscore, E.E., 1993, "Quadratic assignment algorithms for the dynamic layout problem," *International Journal of Production Research*, Vol. 31, No.3, pp. 503-517.

Lawler, E.L., 1963, "The Quadratic Assignment Problem," *Management Science*, Vol. 9, No. 4, pp. 586-599.

Lee, R. and Moore, J.M., 1967, "Corelap-Computerized relationship layout planning," *Journal of Industrial Engineering*, Vol.18, pp. 195-200.

Meller, R.D. and Bozer, Y.A. 1996, "A New Simulated Annealing Algorithm for the Facility Layout Problem," *International Journal of Production Research*, Vol. 34, No. 6, pp. 1675-1692.

Meller, R.D. and Gau, K.Y., 1996a, "The Facility layout problem: Recent and Emerging Trends and Perspectives," *Journal of Manufacturing systems*, Vol. 15, No. 5, pp. 351-366.

Meller, R.D. and Gau, K.Y., 1996b, "Facility layout objective functions and Robust layouts," *International Journal of Production Research*, Vol. 34, No. 10, pp. 2727-2742.

Montreuil, B., 1990, "A Modelling Framework for Integrating Layout Design and Flow Network Design," *Proceedings of the Material Handling Research Colloquium*, Hebron, KY, pp. 43-58.

Montreuil, B. and Laforge.A., 1992, "Dynamic layout design given a scenario tree of probable futures," *European Journal of Operational Research*, Vol. 63, pp. 271-286.

Montreuil, B., Ratliff, H.D., and Goetschalckx, M., 1987, "Matching Based Interactive Facility Layout," *IIE Transactions*, Vol. 19, No. 3, pp. 271-279.

Muther, R. and McPherson, K., 1970, "Four approaches to computerized layout planning," *Industrial Engineering*, February, pp. 39-42.

Neghabat, F., 1974, "An efficient equipment layout algorithm," *Operations Research*, Vol. 22, pp. 622-628.

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

O'Brien, C. and Abdel Barr, S.E.Z., 1980, "An interactive approach to computer aided facility layout," *International Journal of Production Research*, Vol. 18, No. 2, pp. 201-211.

Picone, C.J. and Wilhelm, W.E., 1984, "Perturbation scheme to improve Hillier's solution to the facilities layout problem," *Management Science*, Vol. 30, No. 10, pp. 1238-1249.

Pirlot, M.,1996, "General local search methods," *European Journal of Operational Research*, Vol. 92, pp. 493-511.

Rosenblatt, M.J., 1986, "The dynamics of Plant Layout," *Management Science*, Vol. 32, No. 1, pp. 76-86.

Scriabin, M. and Vergin, R.C., 1985, "A cluster-analytic approach to facility layout," *Management Science*, Vol. 31, No. 1, pp. 33-49.

Seehof, J.M. and Evans, W.O., 1967, "Automated layout design program," *The Journal of Industrial Engineering*, Vol. 18, No. 2, pp. 690-695.

Seppannen, J.J. and Moore, J.M., 1970, "Facilities planning with graph theory," *Management Science*, Vol. 17, No. 4, pp. 242-253.

Seppannen, J.J., and Moore, J.M., 1975, "String processing algorithms for plant layout problems," *International Journal of Production Research*, Vol. 13, No. 3, pp. 239-254.

Skorin-Kopov, J., 1990, "Tabu Search Applied to the Quadratic Assignment Problem," *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 33-45.

Skorin-Kopov, J., 1994, "Extensions of a Tabu Search Adaptation to the Quadratic Assignment Problem," *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 33-45.

Souilah, A., 1995, "Simulated Annealing for the Manufacturing Systems Layout Design," *European Journal Of Operational Research*, Vol. 82, pp. 592-614.

Suresh, G., Vinod, V.V., and Sahu, S., 1995, "A genetic algorithm for facility layout," *International Journal of Production Research*, Vol. 33, No. 12, pp. 3411-3423.

Taillard, E., 1991, "Robust Tabu Search for Quadratic Assignment," *Parallel Computing*, Vol. 17, pp. 443-455.

Tam, K.Y., 1992, "A Simulated Annealing Algorithm for Allocating Space to Manufacturing Cells," *International Journal of Production Research*, Vol. 30, pp. 63-87.

Tate, D.E. and Smith, A.E., 1995a, "A Genetic Approach to the Quadratic Assignment Problem," *Computers and Operations Research*, Vol. 22, pp.73-83.

Tate, D.M. and Smith, A.E., 1995b "Unequal Area Facility Layout Using Genetic Search," *IIE Transactions*, Vol. 27, No. 4, pp. 465-472.

Tompkins, J.A. and Reed, R., 1976, "An applied model for the facilities design problem," *International Journal of Production Research*, Vol. 14, No. 5, pp. 583-595.

Tompkins, J.A., White, J.A., Bozer, Y.A., Frazelle, E.H., Tanchoco, J.M.A., and Trevino, J., 1996, "*Facilities Planning*," NY: John Wiley & Sons, Inc.

Urban, T.L., 1987, "A Multiple Criteria Model for the Facilities Layout Problem," *International Journal of Production Research*, Vol. 25, No.12, pp. 1805-1812.

Urban, T.L., 1993, "A Heuristic for the Dynamic facility layout problem," *IIE Transactions*, Vol.25, No.4, pp. 57-63.

Van Camp, D.J., Carter, M.W., and Vannelli, A., 1991, "A Nonlinear Optimization Approach for Solving Facility Layout Problems," *European Journal of Operational Research*, Vol. 57, pp. 174-189.

Vollman, T.E., Nugent, C.E., and Zartler, 1968, "A computerized model for office layout," *The Journal of Industrial Engineering*, Vol. 19, pp. 321-327.

Wilhelm, M.R. and Ward, T.L., 1987, "Solving the Quadratic Assignment Problems by Simulated Annealing," *IIE Transactions*, Vol. 19, pp. 107-119.

Yip, P.P.C. and Pao, Y.H., 1994, "A guided evolutionary simulated annealing approach to the Quadratic Assignment Problem," *IEEE Transactions*, Vol. 24, No. 9, pp.1383-1387.

Zoller,K. and Adendroff, K., 1972, "Layout Planning by Computer Simulation," *AIIE Transactions*, Vol. 4, No. 2, pp. 116-125.

APPENDICIES

APPENDIX A

Following are the data for the problem instance that is used for the illustrations in section 4.2.5 and in section 4.4.4. The problem instance is taken from Conway and Venkataramanan (1994).

$N = 9$, $T = 5$;

Rearrangement cost = {802, 985, 517, 500, 736, 910, 768, 564, 923};

Flow matrix, $T = 1$;

0	3622	258	493	697	296	627	552	287
991	0	316	443	570	684	334	283	1043
673	6522	0	484	114	324	611	762	762
791	4369	203	0	170	1031	598	923	788
867	5146	56	203	0	1121	309	154	361
894	3264	71	62	769	0	664	343	282
714	3113	240	506	831	1183	0	1144	311
588	1319	319	161	826	1194	744	0	773
1096	6521	335	317	459	439	416	1222	0

Flow matrix, $T = 2$;

0	136	6371	886	1596	213	499	1378	476
657	0	3461	1275	567	254	405	263	449
590	528	0	488	498	273	311	1277	486
179	684	1305	0	1748	101	462	1008	559
772	550	6113	478	0	261	53	1134	1285
511	822	2046	1105	1404	0	384	405	875
577	690	2362	925	944	139	0	847	312
300	461	3343	514	676	128	487	0	214
291	560	6306	397	235	243	466	963	0

Flow matrix, $T = 3$;

0	265	720	3275	361	230	580	221	1433
695	0	816	5276	636	683	637	1877	203
901	1535	0	2322	323	592	129	857	979
1138	298	987	0	400	1051	163	238	924
619	478	856	4205	0	615	81	991	990
647	1373	441	722	608	0	128	603	1040

1008	1383	772	3552	497	836	0	1795	211
1348	682	233	892	206	600	448	0	679
1291	2281	595	3972	89	840	257	348	0

Flow matrix, $T = 4$;

0	753	632	1686	722	241	192	510	63
840	0	897	795	3331	1274	426	611	442
2138	895	0	1277	3019	693	88	470	514
561	445	1444	0	1123	385	523	2015	428
335	421	1549	560	0	820	251	1480	455
636	515	776	1590	5257	0	781	504	416
571	625	765	1304	5312	954	0	647	82
1675	297	176	1137	1240	1313	715	0	321
1187	1550	751	441	840	336	252	1695	0

Flow matrix, $T = 5$;

0	1017	663	1460	1118	804	256	1291	246
854	0	1102	1476	1109	2931	975	1032	403
850	1017	0	1503	412	4102	613	1083	140
525	205	792	0	1060	3647	196	591	981
1653	113	1133	1501	0	2160	203	706	695
981	686	184	852	450	0	155	560	962
781	1010	353	319	648	2043	0	914	185
2031	701	930	755	1113	1883	772	0	175
867	580	377	478	284	4879	106	325	0

Distance matrix (3 x 3 layout)

0	1	2	1	2	3	2	3	4
1	0	1	2	1	2	3	2	3
2	1	0	3	2	1	4	3	2
1	2	3	0	1	2	1	2	3
2	1	2	1	0	1	2	1	2
3	2	1	2	1	0	3	2	1
2	3	4	1	2	3	0	1	2
3	2	3	2	1	2	1	0	1
4	3	2	3	2	1	2	1	0

APPENDIX B

Problem instance PO1 from Lacksonen and Enscore (1993) set of test problems (32 problem instances).

$N = 6$, $T = 3$;

Rearrangement cost = 50;

Because of the symmetry, only lower triangular part of the flow matrix is given;

Flow matrix , $T = 1$;
8
4 2
8 1 4
6 0 8 1
7 3 4 7 5

Flow matrix , $T = 2$;
7
8 1
7 1 1
6 0 7 1
7 5 1 3 6

Flow matrix , $T = 3$;
10
10 1
8 1 1
5 0 4 1
10 1 1 6 5

Distance matrix (2 x 3 layout)
1
2 1
1 2 3
2 1 2 1
3 2 1 2 1

APPENDIX C

Following is the C code for the SA II (reheating) algorithm. The C code for SA I (with out reheating) can be derived from this C code by making the reheating parameter (i.e., *temp_inc*) equal to zero.

SA II ALGORITHM C CODE

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <time.h>
/* by simply making the parameter temp_inc zero, the SA II can be converted into SA I algorithm*/

intN=10,NS=900,factor=1,iter_max=1900,rehe_max=1000,temp_inc=800;/*parameters */
double step=0.99;  /* parameters */
int *p1,*p2,*p3,*p4,*p5,**ar,*re,**pr,*mino,d[30][30],n=30,r_cost=20,jez=0;
int last(),u,v,J=0,r=0,T_per=5,period=0;
long Repcost=0, rep_delta=0, subro(long), total(int t), TFA, *dum, dummy[30][30];
int
w1[30][30],w2[30][30],w3[30][30],w4[30][30],w5[30][30],rep[30],a1[30],a2[30],a3[30],a4[30],a5[30];

FILE *fp,*fp1,*fp2,*fpw1,*fpw2,*fpw3,*fpw4,*fpw5;
int attem_mo=NS*n;

void main()

{
randomize();
clrscr();
int ep=1,swiss,iter=1,iter_no=0,i,reheat=0;
int count,pcount,ncount,no_accp,shilpa;
int *prr[5]={&w1[0][0],&w2[0][0],&w3[0][0],&w4[0][0],&w5[0][0]};
pr=&prr[0];
double pdelf,varx,TI,initial;
```

```
long Min,Min_iter,ramba,Cost,Totalcost=0;

int b,epoch=NN*n;

for(i=0;i<=n-1;i++)  /* initial layout_plan */
  {
   a1[i]=i+1;
   a2[i]=i+1;
   a3[i]=i+1;
   a4[i]=i+1;
   a5[i]=i+1;
  }
p1=&a1[0],p4=&a4[0];
p2=&a2[0],p5=&a5[0];
p3=&a3[0];
int *arr[5]={p1,p2,p3,p4,p5};
ar=&arr[0];
for(i=0;i<=n-1;i++) /*assigning replacement cost to each element of rep[] array*/
rep[i]=r_cost;
re=&rep[0];
dum=&dummy[0][0];
int MinP[200];
mino=&MinP[0];
int buf,j,k,h,x,y,ii;

 if ((fp=fopen("d:\\nilaa\\data\\dist30c.cpp","r")) / * input distance matrix */
     == NULL)
  {
    printf( "Cannot open input file.\n");
    exit (1);
  }


fp2=fopen("d:\\nilaa\\saro\\out.cpp","w"); // output file

fpw1=fopen("d:\\nilaa\\data\\dt305_p1.cpp","r"); /* input: flow matrix T =1; */
fpw2=fopen("d:\\nilaa\\data\\dt305_p2.cpp","r"); /* input: flow matrix T =2; */
fpw3=fopen("d:\\nilaa\\data\\dt305_p3.cpp","r"); /* input: flow matrix T =3; */
fpw4=fopen("d:\\nilaa\\data\\dt305_p4.cpp","r"); /* input: flow matrix T =4; */
fpw5=fopen("d:\\nilaa\\data\\dt305_p5.cpp","r"); /* input: flow matrix T =5; */

for(k=1;k<=n-1;k++)
      {
  for(h=0;h<=k-1;h++)
        {
        fscanf(fp,"%d",&d[h][k]);
```

```
            d[k][h]=d[h][k];
            }
          }
fclose(fp);

for(k=0;k<=n-1;k++)
   {
   d[k][k]=0;
   w1[k][k]=0;
   w2[k][k]=0;
   w3[k][k]=0;
   w4[k][k]=0;
   w5[k][k]=0;
   }

for(x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw1,"%d",&w1[y][x]);
           w1[x][y]=w1[y][x];
        }
        }
fclose(fpw1);
for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw2,"%d",&w2[y][x]);
           w2[x][y]=w2[y][x];
        }
        }
fclose(fpw2);
for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw3,"%d",&w3[y][x]);
           w3[x][y]=w3[y][x];
        }
        }
fclose(fpw3);

for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
```

```
        {
        fscanf(fpw4,"%d",&w4[y][x]);
           w4[x][y]=w4[y][x];
        }
        }
fclose(fpw4);

for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw5,"%d",&w5[y][x]);
           w5[x][y]=w5[y][x];
        }
        }
fclose(fpw5);

/* initialization of dummy array & MinP array */
 for (x=0;x<=n-1;x++)
        {

        for(y=0;y<=n-1;y++)
        {
        dummy[x][y]=0;
        }
           }

  b=0;
  for(k=0;k<=(T_per-1);k++)
     {
      for(i=0;i<=n-1;i++)
         {
          *(mino+b)= *(*(ar+k)+i);
          b=b+1;
         }
     }

/* END*/

b=0;
for(k=0;k<=(T_per-1);k++)
   {
   printf("\n");
   for(i=0;i<=n-1;i++)
          {
         printf(" %d ",*(mino+b));
```

```
        b=b+1;
             }
   }
```

/* Calculation of the total flow cost */

```
for(k=0;k<=T_per-1;k++)
  {
   b=0;
     for(x=0;x<=n-1;x++)
          {
           for(y=0;y<=n-1;y++)
              {
                  dummy[x][y]= *(*(pr+k)+b);
                  b=b+1;

              }
           }

         Cost=total(k);
         Totalcost=Cost+Totalcost;
   }
```

/* Calculation of the total replacement cost */

```
 for(k=0;k<=T_per-2;k++)
    {
     for(x=0;x<=n-1;x++)
          {

           if(( *(*(ar+k)+x))==(*(*(ar+(k+1))+x)) )
              {
           }

           else
              {
              Repcost=Repcost+ *(re+x);

              }

        }
     }

Totalcost=Totalcost+Repcost;
```

```
Min=Totalcost;
Min_iter=Min;
fprintf(fp2," Simulated Annealing applied to DFLP");
fprintf(fp2,"\n");
fprintf(fp2,"\n settings:");
fprintf(fp2,"\n");
fprintf(fp2,"\n temperature factor=%d and step size =%lf",factor,step);
fprintf(fp2,"\n No of attempted moves J =%d and epoch length=%d
",attem_mo,epoch );
fprintf(fp2,"\nMax#ofiterations=%d,rehe_max=%d&

temp_inc=%d",iter_max,rehe_max,temp_inc);
fprintf(fp2,"\n" );
fprintf(fp2,"\n initial assignment");
fprintf(fp2,"\n" );
for(k=0;k<=(T_per-1);k++)
     {
       fprintf(fp2,"\n");
          for(i=0;i<=n-1;i++)
             {
               fprintf(fp2,"%d ",*(*(ar+k)+i));


                  }
     }
fprintf(fp2,"\n Totalcost=%ld",Totalcost);
fprintf(fp2,"\n" );
initial=float(Totalcost)/factor;
time_t first, second;
first = time(NULL);

/*!!!!!!!!!!!!!!!!!!! ALGORITHM STARTS HERE!!!!!!!!!!!!!!!!!!!!!!!!!! */

g1:
   if(reheat = = rehe_max)

      {
       reheat = 0;
       r = r-temp_inc;
      TI = initial*pow(step,(r-1));
      }
   else
     {
       r = r+1;
           TI = initial*pow(step,(r-1));
      }
```

```
 if (Min_iter>Min)
    {
     iter_no = 0;
     reheat = 0;
     Min_iter = Min;
     }
else
   {
     if(iter_no = =iter_max)
          {
           printf("\n Solution Reached");
        printf("\n");
        for(k = 0;k<=(T_per-1);k++)
          {
           printf("\n");
           for(i = 0;i<=n-1;i++)
                 {
                   printf(" %d ", *(*(ar+k)+i) ) ;

                              }
           }
             printf("\n Totalcost = %ld ",Totalcost);
             goto g3;
           }
     else
       {
          iter_no++;
       reheat++;
       }

  }
```

/* Parameter initialization */

```
J=0;ep=1,count=0,pcount=0,ncount=0,no_accp=0;

g2:

ramba = subro(Repcost);

Repcost = Repcost- rep_delta;
J = J+1;

if(ramba = = 0)
        {
          buf = *(*(ar+period)+u);
```

```
            *(*(ar+period)+u) = *(*(ar+period)+v);
            *(*(ar+period)+v) = buf;
          Repcost = Repcost+rep_delta;
          count = count+1;
            swiss = last();
          switch(swiss)
                        {
                        case 1:
                                goto g1;
                        case 2:
                                goto g2;
                         }
        }
else
        {
       if( ramba>0)
                        {
              pcount = pcount+1;
                      ep = ep+1;

                      TFA=Totalcost-ramba;

                          Totalcost = TFA;

                          if(Min>Totalcost)
                            {
                             Min = Totalcost;
                             iter = r;
                       b = 0;
                             for(k = 0;k<=(T_per-1);k++)
                                   {
                         for(i = 0;i<=n-1;i++)
                                        {
                              *(mino+b) = *(*(ar+k)+i);
                              b = b+1;
                                        }
                                   }

                       }

                    if(ep = = epoch)
                              {
                                       goto g1;
                               }
                    else
                              {
```

```
                    swiss = last();
                            switch(swiss)
                              {
                               case 1:

                                    goto g1;
                               case 2:

                                      goto g2;
                              }
                  }

                }

else
                 {


                  pdelf = exp(float(ramba)/TI);
               varx = rand()/32768.0;

                  if(varx<pdelf)
                              {
                               ncount = ncount+1;
                               ep = ep+1;

                               TFA = Totalcost-ramba;

                               Totalcost = TFA;

                               if(Min>Totalcost)
                                   {
                                   Min = Totalcost;
                                   iter = r;
                                    b = 0;
                                   for (k = 0;k<=(T_per-1);k++)
                                           {
                                    for(i = 0;i<=n-1;i++)
                                              {
                                       *(mino+b) = *(*(ar+k)+i);
                                       b = b+1;
                                               }
                                                   }
                               }
                              else
                               {
```

```
                    }

                if(ep = = epoch)
                        goto g1;

                else
                        {
                                swiss = last();
                                        switch(swiss)
                                                {
                                                case 1:
                                                        goto g1;
                                                case 2:
                                                                goto g2;
                                                }


                        }


                }
        else
                {

                        buf = *(*(ar+period)+u);
                        *(*(ar+period)+u)  = *(*(ar+period)+v);
                        *(*(ar+period)+v) = buf;
                Repcost = Repcost+rep_delta;
                no_accp = no_accp+1;
                                swiss = last();
                                switch(swiss)
                                  {
                                   case 1:
                                                        goto g1;
                                        case 2:
                                                goto g2;
                                        }
                }

                  }


}


g3:
```

```
b=0;
for(k = 0;k<=(T_per-1);k++)
  {
   fprintf(fp2,"\n");
   for(i = 0;i<=n-1;i++)
           {
        fprintf(fp2,"%d ",*(mino+b));
           b = b+1;
           }
   }
second = time(NULL);

fprintf(fp2,"\n Solution reached in %f minutes ",difftime(second,first)/60.0);
fprintf(fp2,"\n Minimum cost= %ld found at the iteration=%d",Min,iter);
fclose(fp2);
return;

}

/ * SUBROUTINES * /

long subro( long r_cost)  /* subroutine for pairwise exchange */
{

int i,j,l,f,mo,zo,dtc,b,buf,x,y,k;
long calc,diff,imp,costy = 0,delta = 0;
     period = random(T_per);
     b = 0;
     for(i = 0;i<=n-1;i++)
          {
           for(j = 0;j<=n-1;j++)
             {
              dummy[i][j] =  *(*(pr+period)+b);
              b = b+1;
             }
       }

do
      {
          i = random(n);
          j = random(n);
       }while(i = = j);

         zo = *(*(ar+period)+i);
        mo = *(*(ar+period)+j);
```

```
        diff = 0;
        for(l = 0;l<=n-1;l++)
             {
             f = *(*(ar+period)+l);
                //dtc = (dummy[i][l]-dummy[l][j]-dummy[j][l]+dummy[l][i])*(d[f-
                                    1][zo-1]-d[f- 1][mo-1]); /* asymmetric */
                dtc = (dummy[l][i]-dummy[l][j])*(d[f-1][zo-1]-d[f-1][mo-1]);
                                                                    /*symmetric*/
               diff = diff+dtc;

             }

                calc = 2*dummy[i][j]*d[zo-1][mo-1]; /* symmetric */
               //calc =  2*d[zo-1][mo-1]*(dummy[i][j]+dummy[j][i]); /* asymmetric
*/
               delta = diff-calc;

u = i;
v = j;
buf = *(*(ar+period)+u);
*(*(ar+period)+u) = *(*(ar+period)+v);
*(*(ar+period)+v) = buf;


for(k = 0;k<=T_per-2;k++)
    {
     for(x = 0;x<=n-1;x++)
            {

             if(( *(*(ar+k)+x)) = = (*(*(ar+(k+1))+x)) )
                {
                 }
            else
                 costy = costy+ *(re+x);
             }
    }

rep_delta = r_cost-costy; /* old repcost(r_cost) - new repcost(cost) */
imp = delta+rep_delta;
jez++;
return(imp);
}

long total ( int t)  /* subroutine for total cost * and the value of period is
passed to t */
{int z,m,j,k;
```

```
long TC = 0,prod;
    for(j = 0;j<=n-2;j++)
        {
           for(k = j+1;k<=n-1;k++)
           {
              z = *(*(ar+t)+j);
              m = *(*(ar+t)+k);
                    //prod = (dummy[j][k]+dummy[k][j])*d[z-1][m-1]; /*
                                                          asymmetric*/
              prod = dummy[j][k]*d[z-1][m-1]; / *symmetric */
              TC  = TC+prod;
           }
           }
return (TC);
}


int last( )  /* subroutine to check the number of attempted exchanges in an
epoch */
      {
      int varg = 0;
               if(J>attem_mo)
                        varg = 1;

                else
                        varg = 2;
      return (varg);
      }
```

SA COMBO ALGORITHM C CODE

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<dos.h>
#include<time.h>

int NN=10,NS=500,iter_max=1000,rehe_max=100,temp_inc=0; /* parameters*/
int step = 0.95; /* parameters*/
int*p,*p1,*p2,*p3,*p4,*p5,**ar,*re,**pr,*mino,*ap,d[30][30],n=30,r_cost= 0,jez
= 0;
int last(),u,v,u_b,v_b,delta,IT,J=0,r=1,R=500,T_per =
                                                    5,period,peri,window,rey[30];
int w1[30][30],w2[30][30],w3[30][30],w4[30][30],w5[30][30];
longsubro(),total(),total_b(),total1(int),subro_b(),total_b1(int),*dum,dummy[30][
                                                                        30];
int  rey_b[30], rey_b1[30], dummy_b[30][30],back[5][30];
FILE *fp,*fp1,*fp2,*fpw1,*fpw2,*fpw3,*fpw4,*fpw5;

int attem_mo = NS*n;
void main()
{
randomize();
clrscr();
int ep = 1,swiss,swiss1,finalcost,iter = 1,big,big1,big2,t,t_b,reheat = 0;
int count = 0,pcount = 0,ncount = 0,no_accp = 0,shilpa,ini_te,iter_no = 0,i;

double pdelf, varx;
double TI, initial;
long
TFA,Cost,Repcost=0,Totalcost=0,Totalcost_b=0,Repcost_b=0,ramba,ramba_b,gil
ma_b,sum_b,Min,Min_iter;
int pair_no = n*(n-1)/2,bim;
int b,epoch = 10*n;
int a[30];

for(i = 0;i<=n-1;i++) /* initial layout */
a[i] = i+1;

p = &a[0];
int abb[30];
int a1[30],a2[30],a3[30],a4[30],a5[30],ape[30];
p1 = &a1[0],p4 = &a4[0];
```

```
p2 = &a2[0],p5 = &a5[0];
p3 = &a3[0];
int *arr[5]={p1,p2,p3,p4,p5};
ar = &arr[0];
int rep[30];
for(i = 0;i<=n-1;i++)
rep[i] = r_cost;
re = &rep[0];
dum=&dummy[0][0];
int *prr[5] = { &w1[0][0],&w2[0][0],&w3[0][0],&w4[0][0],&w5[0][0]};
pr = &prr[0];
int MinP[100];
mino=&MinP[0];
long Costy[5];
int buf,j,k,h,x,y,ii,o=0;

 if ((fp=fopen("d:\\nilaa\\data\\dist30c.cpp","r")) /* distance matrix input */
     == NULL)
  {
    printf( "Cannot open input file.\n");
    exit (1);
  }
fp2 = fopen("d:\\nilaa\\saro\\out.cpp","w");  / * output file */
fpw1 = fopen("d:\\nilaa\\data\\dt307_p1.cpp","r"); /* input flow matrix for T =
1*/
fpw2 = fopen("d:\\nilaa\\data\\dt307_p2.cpp","r"); /* input flow matrix for T =
2*/
fpw3 = fopen("d:\\nilaa\\data\\dt307_p3.cpp","r"); /* input flow matrix for T =
3*/
fpw4 = fopen("d:\\nilaa\\data\\dt307_p4.cpp","r"); /* input flow matrix for T =
4*/
fpw5 = fopen("d:\\nilaa\\data\\dt307_p5.cpp","r"); /* input flow matrix for T =
5*/

for(k = 1;k<=n-1;k++)
        {
         for(h = 0;h<=k-1;h++)
           {
           fscanf(fp,"%d",&d[h][k]);
              d[k][h] = d[h][k];
              }
            }
fclose(fp);

for(k= 0;k<=n-1;k++)
   {
```

```
   d[k][k]  = 0;
   w1[k][k] = 0;
   w2[k][k] = 0;
   w3[k][k] = 0;
   w4[k][k] = 0;
   w5[k][k] = 0;
   }


for(x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw1,"%d",&w1[y][x]);
           w1[x][y]=w1[y][x];
        }
        }
fclose(fpw1);
for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw2,"%d",&w2[y][x]);
           w2[x][y]=w2[y][x];
        }
        }
fclose(fpw2);
for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw3,"%d",&w3[y][x]);
           w3[x][y]=w3[y][x];
        }
        }
fclose(fpw3);

for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw4,"%d",&w4[y][x]);
           w4[x][y]=w4[y][x];
        }
        }
fclose(fpw4);
```

```
for (x=1;x<=n-1;x++)
        {
        for(y=0;y<=x-1;y++)
        {
        fscanf(fpw5,"%d",&w5[y][x]);
           w5[x][y]=w5[y][x];
        }
        }
fclose(fpw5);
fprintf(fp2," SA with forecasting windows & Backward Pairwise");
fprintf(fp2,"\n");
fprintf(fp2,"\n settings:");
fprintf(fp2,"\n");
fprintf(fp2,"\n initial temperature=%lf and step size =%lf",initial,step);
fprintf(fp2,"\n No of attempted moves J =%d and epoch length=%d
",attem_mo,epoch );
fprintf(fp2,"\n Max # of iterations=%d",R);
fprintf(fp2,"\n" );
fprintf(fp2,"\n initial assignment");
fprintf(fp2,"\n" );
 for (x=0;x<=n-1;x++)
       {
          fprintf(fp2," %d ",a[x]);
          ape[x]=a[x];
       }
fprintf(fp2,"\n" );

/* initialization of dummy array and rey array*/
 for (x=0;x<=n-1;x++)
        {
        for(y=0;y<=n-1;y++)
        {
        dummy[x][y]=0;
        }
           }
  for (x=0;x<=n-1;x++)
          rey[x]=0;
time_t first, second;
   first = time(NULL);

/* !!!!!!!!!!SA  COMBO ALGORITHM  STARTS   HERE!!!!!!!!!!!!!!!!!!!*/

for (window = 0;window<=T_per-1;window++)
   {
         for(period = 0;period<=T_per-1;period++)
```

```
            {
                if(period<=(T_per-1-window))
                     {
                      big1 = period;
                  big2 = period+window;
                  }
                 else
                  {
                   big1 = period;
                   big2 = T_per-1;
                   }
                 for(big = big1;big<=big2;big++)
                  {
                  b = 0;
                   for(x = 0;x<=n-1;x++)
                     {
                      for(y = 0;y<=n-1;y++)
                            {
                             dummy[x][y] = dummy[x][y]+ *(*(pr+big)+b);
                             b = b+1;
                            }

                   }

               }
Cost = total();
Min = Cost;
Min_iter = Cost;
for(i = 0;i<=n-1;i++)
   {
    MinP[i] = a[i];
   }

initial = float(Cost);
printf("\n initial=%f",initial);

g1:

 if(reheat==rehe_max)
     {
     reheat=0;
     r=r-temp_inc;
     }
 r=r+1;
 TI=initial*pow(step,(r-1));
```

```
if(Min_iter > Min)
   {
   iter_no=0;
   Min_iter=Min;
   }
else
   {
    if(iter_no==iter_max)
          {
          printf("\n STOP");
          goto g3;
      }
    else
      {
          iter_no=iter_no+1;
       reheat = reheat+1;
          }
    }
```

/* Parameter initialization */

```
J=0;ep=1,count=0,pcount=0,ncount=0,no_accp=0;
g2:
ramba= subro();
J=J+1;
if(ramba==0)
        {
            count = count+1;
            swiss = last();
         switch(swiss)
                        {
                        case 1:
                                goto g1;
                        case 2:
                                goto g2;
                         }
           }
else
        {
           if( ramba>0)
                {
                pcount = pcount+1;
                      ep = ep+1;
                TFA = Cost-ramba;
                      Cost = TFA;
                      /* updating the rey array */
```

```
            if(rey[u]>0 && rey[v]>0)
              {
              }
            else
        {
               if(rey[u = =0 && rey[v]>0)
                     {
                      rey[u] = 1;
                     }
              else
            {
                     if(rey[u]>0 && rey[v]= = 0)
                          {
                          rey[v] = 1;
                          }
                      else
                          {
                          rey[u] = 1;
                          rey[v] = 1;
                          }
                      }
                  }

            t = *(u+p);
      *(u+p) = *(v+p);
            *(v+p) = t;
if(Min>Cost)
              {
               Min = Cost;
               iter = r;

               for(I = 0;i<=n-1;i++)
            {
              MinP[i] = a[i];
            }

        }
    if(ep = = epoch)
              {
                       goto g1;

              }
      else
              {
                     swiss = last();
                              switch(swiss)
```

```
                                        {
                                         case 1:

                                              goto g1;
                                         case 2:

                                                goto g2;
                                        }


                    }
                 }
else
        {
              ncount = ncount+1;
              pdelf = exp(float(ramba)/TI);
             varx = rand()/32768.0;

              if(varx<pdelf)
                            {

                                ep = ep+1;
                                TFA = Cost-ramba;
                                Cost = TFA;
                                /* updating the rey array */
                          if(rey[u]>0 && rey[v]>0)
                             {
                             }
                          else
                            {
                                if(rey[u] = = 0 && rey[v]>0)
                                        rey[u] = 1;

                                 else
                                      {
                                        if(rey[u]>0 && rey[v]==0)
                                             rey[v]=1;
                                         else
                                            {
                                             rey[u]=1;
                                             rey[v]=1;
                                            }
                                      }
                                  }
                                t = *(u+p);
                      *(u+p) = *(v+p);
```

```
                    *(v+p) = t;

                if(ep = = epoch)
                    goto g1;

                  else
                    {



                            swiss=last();
                                switch(swiss)
                                {
                                case 1:
                                    goto g1;
                                case 2:
                                        goto g2;
                                }
                            }
                    }
                else
                    {
                            no_accp=no_accp+1;
                             swiss=last();
                             switch(swiss)
                               {
                                case 1:
                                        goto g1;
                                    case 2:
                                        goto g2;
                                    }
                    }
                 }
}
g3:

    for(i=0;i<=n-1;i++)
        *(*(ar+period)+i)=MinP[i];

    for (x=0;x<=n-1;x++)  /*at every time, when period ends it dummy array& rey array
                                                has to be initialized to zero*/
        {
      for(y=0;y<=n-1;y++)
            dummy[x][y]=0;
```

```
        }

  for (x=0;x<=n-1;x++)
       rey[x]=0;

      r=0;
      iter_no=0;
    reheat=0;

 }/* PERIOD LOOP Ends */

      for (x=0;x<=n-1;x++) /*dummy array has to be initialized to zero */
    {
     for(y=0;y<=n-1;y++)
      {
      dummy[x][y]=0;
      }
       }
      for(x=0;x<=T_per-1;x++)/* Costy array need to be initialized*/
   Costy[x]=0;
      for(x=0;x<=n-1;x++)
      a[x]=ape[x];

      Totalcost=0;/*Totalcost and Repcost are intialized*/
      Repcost=0;
      fprintf(fp2,"\n window=%d ",window);

          for(k=0;k<=T_per-1;k++)  /* calculation of the flow cost for a
window */
              {
                 b=0;
                 for(x=0;x<=n-1;x++)
                       {
                        for(y=0;y<=n-1;y++)
                          {
                               dummy[x][y]= *(*(pr+k)+b);
                               b=b+1;
                        }
                         }
                           Costy[k]=total1(k);
                           Totalcost=Costy[k]+Totalcost;

              }

                for(k=0;k<=(T_per-1);k++)
                    {
```

```
                fprintf(fp2,"\n");
                for(i=0;i<=n-1;i++)
                   {
                     fprintf(fp2,"%d ", *(*(ar+k)+i));

                      }
               }
             for(k=0;k<=T_per-2;k++) /* calculation of the replacement cost
                                                    for a    window */
                {
               for(x=0;x<=n-1;x++)
                      {

                      if(( *(*(ar+k)+x))==(*(*(ar+(k+1))+x)) )

                      else
                         Repcost=Repcost+ *(re+x);

                    }
                 }

               fprintf(fp2,"\n Totalcost =%ld ",(Totalcost+Repcost));
               fprintf(fp2,"\n");

                for (x=0;x<=n-1;x++) /*dummy array has to be initialized to
                                                              zero*/
              {
              for(y=0;y<=n-1;y++)
                {
                       dummy[x][y]=0;
                }
                      }
/* !!!!!!!!!!!BACKWARD PASS WISE EXCHANGE!!!!!!!!!!!!!!!!!!!*/


                    for(x = 0;x<=n-1;x++)   /* abb[] is initialized to last
period's
                                                              assigment */
                     {
                     back[T_per-1][x] = *(*(ar+T_per-1)+x);  /* last period
layout is
                                                    assigned to last row of back
array*/
                     }

               for(peri = (T_per-2);peri>=0;peri--)
```

```
    {
        for(x = 0;x<=n-1;x++)
        abb[x]=*(*(ar+peri)+x); /* T-1 period's assignment
                                        is taken*/

        ap = &abb[0];       /* pointer assigned to the abb []
                                        array* /

        b = 0;
for(x=0;x<=n-1;x++)/* assigning weight matrix to
                                    dummy_b array*/
    {
     for(y = 0;y<=n-1;y++)
          {
           dummy_b[x][y] =  *(*(pr+peri)+b);
           b=b+1;
          }

           }

sum_b = total_b();
            ramba_b = subro_b();

            while (ramba_b>0)
        {
     sum_b = sum_b-ramba_b;
           ramba_b = 0;

     /* updating the rey_b[] and rey_b1[] array*/
                  if(peri = = 0)
                    {
                     if(*(*(ar+peri+1)+u_b) != *(ap+v_b))
                                /*replacement of dept i
                                        ,previous*/
                          rey_b[u_b] = 1;
                else
                          rey_b[u_b] = 0;

                       if(*(*(ar+peri+1)+v_b) !=
                                        *(ap+u_b))
                             /*replacement of dept j
                                        ,previous*/
                          rey_b[v_b] = 1;
                else
                          rey_b[v_b] = 0;
```

```
                }
              else
{

                  if( *(*(ar+peri-1)+u_b) != *(ap+v_b))
                                /*replacement of dept i
                                                ,next*/
                        rey_b1[u_b] = 1;
              else
                        rey_b1[u_b] = 0;

            if( *(*(ar+peri+1)+u_b) != *(ap+v_b) )
                                /*replacement of dept i
                                                ,previous*/
                        rey_b[u_b] = 1;
 else
                        rey_b[u_b] = 0;

              if( *(*(ar+peri-1)+v_b) != *(ap+u_b) )
                                /*replacement of dept j
                                                ,next*/
                        rey_b1[v_b] = 1;
              else
                        rey_b1[v_b] = 0;

              if( *(*(ar+peri+1)+v_b) != *(ap+u_b) )
                                /*replacement of dept j
                                                ,previous*/
                        rey_b[v_b] = 1;
              else
                        rey_b[v_b] = 0;

    }

     t_b = *(u_b+ap);
*(u_b+ap) = *(v_b+ap);
*(v_b+ap ) = t_b;

     gilma_b = subro_b();
     ramba_b =  ramba_b+gilma_b;
              }
for(i=0;i<=n-1;i++)
              {
              back[peri][i] = abb[i];
              *(*(ar+peri)+i) = abb[i];/* ar array is
                                        updated*/
```

```
        }

for (x = 0;x<=n-1;x++) /*at every time , when perid_b ends it
  dummy array& rey array has to be initialized to zero*/
          {
                        rey_b[x] = 0;
                        rey_b1[x] = 0;
           for(y = 0;y<=n-1;y++)
             {
                          dummy_b[x][y] = 0;
              }
             }


        } /* peri loop ends here*/

        for (x = 0;x<=n-1;x++) /*dummy array has to be
                                          initialized to zero*/
           {
  for(y = 0;y<=n-1;y++)
     dummy[x][y] = 0;
           }
   for(x = 0;x<=T_per-1;x++)/* Costy array need to be
                                              initialized*/
Costy[x] = 0;
    Totalcost_b = 0;/*Totalcost and Repcost are intialized*/
    Repcost_b = 0;
           for(k = 0;k<=T_per-1;k++) /* calculation of the
                                              Totalcost */
     {
      b = 0;
      for(x = 0;x<=n-1;x++)
           {
           for(y = 0;y<=n-1;y++)
             {
                dummy_b[x][y] =  *(*(pr+k)+b);
                b = b+1;
                     }
                }


              Costy[k] = total_b1(k);
              Totalcost_b = Costy[k]+Totalcost_b;

     }
```

```
        for(k = 0;k<=T_per-2;k++)
                        {
                        for(x = 0;x<=n-1;x++)
                                {
                                 if( back[k][x] = = back[k+1][x] )
                                   {
                                                }
                                    else
                                   {
                                  Repcost_b = Repcost_b+ *(re+x);

                                   }

                               }
                             }


                       for(k=0;k<=(T_per-1);k++)
                                        {
                                         printf("\n");
                                         fprintf(fp2,"\n");
                          for(i=0;i<=n-1;i++)
                                {
                               fprintf(fp2,"%d ",back[k][i]);
                                        }
                                      }
                               fprintf(fp2,"\n Totalcost_b =%ld
                                              ",Totalcost_b+Repcost_b);

                    for(x=0;x<=T_per-1;x++)
                              Costy[x]=0;

        } /* WINDOW LOOP ENDS HERE*/

second = time(NULL);

fprintf(fp2,"\n The total time is %f minutes \n", difftime(second,first)/60.0);
printf("\n THE END");
fclose(fp2);

return;

}
```

/*SUBROUTINES*/

```
long subro()  /* subroutine for pairwise exchange*/
{
int i,j,l,f,mo,zo,b,x,y,k;
long delta=0,imp,diff,dtc;

     do
      {
          i = random(n);
          j = random(n);
       }while(i = = j);

        zo = *(p+i);
        mo = *(p+j);
        diff = 0;
            for(l = 0;l<=n-1;l++)
              {
                 f = *(l+p);
                 //dtc=(dummy[i][l]-dummy[l][j]-dummy[j][l]+dummy[l][i])*(d[f-
                                        1][zo-1]-d[f-1][mo-1]); /* asymmetric */
                 dtc = (dummy[l][i]-dummy[l][j])*(d[f-1][zo-1]-d[f-1][mo-1]); /*
                                                                    symmetric */
              diff=diff+dtc;

              }
                 //delta=diff-2*d[zo-1][mo-1]*(dummy[i][j]+dummy[j][i]); /*
                                                          asymmetric */
                 delta=diff-2*dummy[i][j]*d[zo-1][mo-1]; /* symmetric */

u=i;
v=j;
     if(period = = 0)
     {
     imp = delta;
     goto E1;
     }

     if(rey[u] = = 0 && rey[v] = = 0)
          imp = delta-*(re+u)-*(re+v);

    else
      {
      if(rey[u]>0 && rey[v] = = 0)
          imp = delta- *(re+v);

      else
```

```
            {
             if(rey[u] = = 0 && rey[v]>0)
            imp = delta- *(re+u);
             else
            imp = delta;

        }
      }

E1:
jez++;

return(imp);
}
```

long total( ) **/* subroutine for total cost and only once it is called in */**

```
{
int z,m,hh,qq;
long TC = 0,prod;
     for(hh = 0;hh<=n-2;hh++)
          {
             for(qq = hh+1;qq<=n-1;qq++)
                {
                      z = *(hh+p);
                m = *(qq+p);
                      //prod = (dummy[hh][qq]+dummy[qq][hh])*d[z-1][m-1];/*
                                                              asymmetric */
                      prod = dummy[hh][qq]*d[z-1][m-1];       /* symmetric */
                      TC = TC+prod;

              }
             }
return(TC);
}


int last()
    {
      int varg = 0;

      if(J>attem_mo)
            varg = 1;
```

```
        else
            varg = 2;

         return(varg);
    }

long total1(int t) /* subroutine for total cost */

{                   /* value of period is passed to t* /
int z,m,j,k;
long prod,TC = 0;

    for(j = 0;j<=n-2;j++)
        {
           for(k = j+1;k<=n-1;k++)
           {
              z = *(*(ar+t)+j);
              m = *(*(ar+t)+k);
                    //prod = (dummy[j][k]+dummy[k][j])*d[z-1][m-1];
                                                          /*asymmetric/
                    prod = dummy[j][k]*d[z-1][m-1];   /*symmetric */
              TC = TC+prod;
            }
           }

return(TC);
}

/*!!!!!!!!!! subroutines for the backward pass algorithm !!!!!!!!!!!!!!!!*/

long subro_b( )  /* subroutine for backward pairwise exchange*/
{
int i,j,l,f,mo,zo;
int repi,repj,repi_b,repj_b,x;
long e,diff, delta = 0,imp,dtc;
e = 0;
mo = 0;
zo = 0;
for(i = 0;i<=n-2;i++)
          {
          for(j  = i+1;j<=n-1;j++)
              {
                    zo = *(i+ap);
                    mo = *(j+ap);
                              diff = 0;
                              for(l = 0;l<=n-1;l++) /* to find out the delta for each
```

```
                                                    pairwise  exchange*/
                    {
                        f=*(l+ap);
                        dtc = (dummy_b[l][i]-dummy_b[l][j])*(d[f-1][zo-
                                    1]-d[f-1][mo-1]); /* symmetric/
                        //dtc=(dummy_b[i][l]-dummy_b[l][j]-
dummy_b[j][l]+dummy_b[l][i])*(d[f-1][zo-1]-d[f-1][mo-1]); /*
                                                    asymmetric */
                  diff=diff+dtc;
                }
                        delta=diff-2*dummy_b[i][j]*d[zo-1][mo-1]; /*
                                                    symmetric */
                       //delta=diff-2*d[zo-1][mo-
                            1]*(dummy_b[i][j]+dummy_b[j][i]); /*
                                                    asymmetric */


                 if(peri = = 0)
                {
                        repi_b = 0;/* next period is not considered */
                        repj_b = 0;/* next period is not considered */
        if(*(*(ar+peri+1)+i) != *(ap+j) ) /*replacement of dept i
                                                    ,previous */
                        repi = 1;
                    else
                        repi = 0;

        if(*(*(ar+peri+1)+j) != *(ap+i) ) /*replacement of dept
                                                    j , previous */
                         repj = 1;
                     else
                         repj = 0;

                         goto tt1;
                 }
         else
                        {
                 }


        if( *(*(ar+peri-1)+i) != *(ap+j) ) /*replacement of dept i ,next*/
                    repi_b = 1;
                 else
                    repi_b = 0;

        if( *(*(ar+peri+1)+i) != *(ap+j) )/*replacement of dept i
```

```
                                                    ,previous*/
            repi = 1;
        else
            repi = 0;

if( *(*(ar+peri-1)+j) != *(ap+i) ) /*replacement of dept j ,next*/
            repj_b = 1;
        else
            repj_b = 0;

if( *(*(ar+peri+1)+j) != *(ap+i) ) /*replacement of dept j
                                                    ,previous*/
            repj = 1;
        else
            repj = 0;



  tt1:

if(repi = = 1)/* decision taken to include the replacment cost*/
           {
            if(rey_b[i]>0)
                repi = 0;/* decision re evaluated and rep cost is not
                                                        included*/
            else
               repi = 1;

            }

if(repi_b = =1)/* decision taken to include the replacment cost*/
           {
 if(rey_b1[i]>0)
          repi_b = 0;/* decision re evaluated and rep cost is not
                                                        included*/
            else
                 repi_b = 1;
            }

        if(repj = = 1)/* decision taken to include the replacment
                                                        cost*/
           {
            if(rey_b[j]>0)
                repj = 0;/* decision re evaluated and rep cost is not
                                                        included*/
            else
```

```
                repj = 1;
              }
            if(repj_b = = 1)/* decision taken to include the replacment
                                                                cost*/
              {
               if(rey_b1[j]>0)
                    repj_b = 0;/* decision re evaluated and rep cost is
                                                    not   included*/
               else
                    repj_b = 1;
            }



          imp = delta-*(re+i)*(repi+repi_b)-*(re+j)*(repj+repj_b);

        if(imp>e)
              {
               e = imp;
               u_b = i;
               v_b = j;

               }

      }/* for j loop ends*/


   }/*for i loop ends*/

return(e);

}


long total_b() /* subroutine for the total cost */
{
 int z,m,i,j;
 long TC=0,prod;

 for(i = 0;i<=n-2;i++)
        {
        for(j = i+1;j<=n-1;j++)
              {
              z = *(i+ap);
              m = *(j+ap);
          //prod = (dummy_b[i][j]+dummy_b[j][i])*d[z-1][m-1]; /*
```

```
                                                              asymmetric */
                prod = dummy_b[i][j]*d[z-1][m-1]; /* symmetric */
                   TC = TC+prod;
                   }
          }

 return(TC);

 }

long total_b1(int t)  /* subroutine for total cost */

{                    /* value of period is passed to t */
int z,m,j,k;
long TC = 0,prod;

     for(j = 0;j<=n-2;j++)
            {
            for(k=j+1;k<=n-1;k++)
            {
               z = back[t][j];
              m = back[t][k];
               //z = *(*(ar+t)+j);
               //m = *(*(ar+t)+k);
                     //prod = (dummy_b[j][k]+dummy_b[k][j])*d[z-1][m-1]; /*
                                                              asymmetric */
                     prod = dummy_b[j][k]*d[z-1][m-1];       /* symmetric */
               TC = TC+prod;
             }
            }

return(TC);
}
```