HIERARCHICAL MANAGEMENT OF AN INTEGRATED PRODUCTION-LOGISTICS SYSTEM

by

Demet Özgür

B.S. in Industrial Engineering, Boğaziçi University, 1995

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Industrial Engineering

Boğaziçi University

1997

HIERARCHICAL MANAGEMENT OF AN INTEGRATED PRODUCTION-LOGISTICS SYSTEM

APPROVED BY:

Assoc.Prof. Gülay Barbarosoğlu .....
(Thesis Supervisor)

Prof.Gündüz Ulusoy .....

Assoc. Prof. Bülent Durmuşoğlu .....

DATE OF APPROVAL 13/6/1997

To my parents...

Vicdan ÖZGÜR

Necati ÖZGÜR

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my thesis supervisor, Assoc. Prof. Gülay BARBAROSOĞLU, for her invaluable guidance and advice, encouragement and motivation not only during this thesis but also throughout my graduate study.

I would like to thank to Prof. Gündüz ULUSOY and Assoc. Prof. Bülent DURMUŞOĞLU for their valuable comments and suggestions as members of my thesis committee.

I am very thankful to Murat ÖZDEMİR, my lab partner, for his valuable help, moral support, encouragement and kindness during this thesis. I also wish to thank to my friend Mine TEZER for her suggestions, interest and continuous moral support throughout all phases of this study.

Finally, I am deeply grateful to my family for their invaluable support, patience and never ending encouragement not only during this research, but at all stages of my life.

ABSTRACT

This study analyzes the production-logistics system of a manufacturing environment, where a manufacturer distributes its products to customers via multi-depots, within the framework of hierarchical planning. The production-logistics system is decomposed into two levels; aggregate production-distribution decisions are handled in the first level while vehicle routing decisions are addressed in the second level. The first level problem is further decomposed into production and distribution subproblems by using Lagrangean relaxation. Exact and heuristic algorithms are developed for these subproblems, and a sequential structure is designed to maintain consistency and feasibility between these solutions. Then the result of this level is downloaded to the vehicle routing level where a tabu search heuristic is developed and implemented. Results reveal the importance of coordination and information exchange in a hierarchical plan upon the overall performance of the integrated production-logistics system.

ÖZET

Bu çalışma, bir imalatçının ürünlerini birden çok depo vasıtasıyla müşterilerine dağıttığı bir imalat ortamının üretim-lojistik sistemini hiyerarşik bir planlama çerçevesinde inceler. Üretim-lojistik sistemi iki seviyeye ayrıştırılır; toplaşık üretim-dağıtım kararları ilk seviyede ele alınırken araç rotalama kararları ikinci seviyede çözülür. İlk seviye problemi de Lagrange çarpanlarının kullanıldığı bir çözüm yöntemiyle üretim ve dağıtım alt problemlerine ayrıştırılır. Bu problemlerin çözümleri arasında tutarlılık ve fizibilite elde etmek için sıralı bir yapı tasarlanmıştır. Bu seviyenin sonuçları bir yasaklı arama meta-sezgiselinin geliştirilip uygulandığı araç rota belirleme seviyesine verilir. Sonuçlar, hiyerarşik bir plan içindeki koordinasyon ve bilgi değişiminin önemini, bütünsel üretim-lojistik sisteminin tüm performansı üzerinde gösterir.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iv

ABSTRACT v

ÖZET vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

LIST OF TABLES x

1. INTRODUCTION 1

2. LITERATURE SURVEY 5

3. PROBLEM DEFINITION 16

4. HIERARCHICAL DESIGN OF THE INTEGRATED PRODUCTION-LOGISTICS SYSTEM 20

4.1. Integrated Production-Logistics Model 21

4.2. Hierarchical Decomposition of the Production-Logistics System 27

4.2.1. Aggregate Production-Distribution Model 28

4.2.2. Vehicle Routing Problem 31

4.3. Organizational Design 32

5. LAGRANGEAN RELAXATION APPROACH TO THE AGGREGATE PRODUCTION AND DISTRIBUTION PROBLEM 34

5.1. Construction of the Lagrangean Relaxation Technique 34

5.2. Relaxed Production and Distribution Problem 37

5.2.1. Solution Procedure for Production Submodel 38

5.2.2. Solution Procedure for Distribution Problem 43

5.3. Subgradient Optimization Procedure 48

5.4. The Lagrangean Heuristics 54

5.5. Interpretation of Dual Varibels 62

6. A TABU SEARCH ALGORITHM FOR THE VEHICLE ROUTING PROBLEM .. 64

6.1. Problem Formulation 64

6.2. Tabu Search Algortithm 67
6.2.1. Construction of an Initial Solution 71
6.2.2. The Insertion Procedure 72
6.2.3. The 2-Opt Improvement Procedure 72
6.2.4. Intensification Strategy 73
6.3. Computational Results 78
7. EXPERIMENTAL DESIGN AND COMPUTATIONAL RESULTS 81
8. CONCLUSION 88
APPENDIX A: THE RESULTS OF COMPUTATIONAL STUDY 92
APPENDIX B: LAGRANGEAN RELAXATION, DISTAL, DISTFEAS, PROFEAS CODES 102
APPENDIX C: DETABA AND TANE CODES 136
REFERENCES 170

LIST OF FIGURES

		Page
FIGURE 3.1	The production-logistics system of a manufacturing environment.	18
FIGURE 3.2	The Lagrangean relaxation information flow.	19
FIGURE 3.3	The Lagrangean heuristic information flow.	19
FIGURE 4.2.1	Hierarchical decomposition of the production-logistics system.	27
FIGURE 4.3.1	Parallelism between business funtions and organizational units.	33
FIGURE 5.2.2.1	Flow chart of DISTAL.	44
FIGURE 5.3.1	Subgradient optimization procedure to solve problem L.	50
FIGURE 5.4.1	Flow chart of DISTFEAS.	60
FIGURE 5.4.2	Flow chart of PROFEAS.	61
FIGURE 6.2.1	Flow chart of TANE.	70
FIGURE 6.2.2	Flow chart of solution improvement	75
FIGURE 6.2.3	Flow chart of neighbourhood search	76
FIGURE 6.2.4	Flow chart of DETABA.	77
FIGURE 7.1	Test problem generation procedure.	84
FIGURE 7.2	A possible vehicle routing solution for (j,t)=(1,1).	85

LIST OF TABLES

		Page
TABLE 6.4.1	Objective function values in each DETABA phase for the test problems.	78
TABLE 6.4.2	Computational comparison of various TS algorithms.	80
TABLE 7.1	Characteristics of data sets.	80
TABLE A.6.4.3	Best VRP solution results for the test problem 1.	93
TABLE A.6.4.4	Best VRP solution results for the test problem 2.	93
TABLE A.6.4.5	Best VRP solution results for the test problem 3.	93
TABLE A.6.4.6	Best VRP solution results for the test problem 4.	94
TABLE A.6.4.7	Best VRP solution results for the test problem 5.	94
TABLE A.6.4.8	Best VRP solution results for the test problem 6.	95
TABLE A.6.4.9	Best VRP solution results for the test problem 7.	95
TABLE A.7.2	Computational results of DISTAL.	96
TABLE A.7.3	Computational results of the Lagrangean relaxation algorithm.	97
TABLE A.7.4	VRP solutions of data set 9.	99

1. INTRODUCTION

Today in a dynamic global economy and competitive market environment, it is becoming extremely critical to respond and adapt quickly to changes. The traditional roles of marketing, sales, logistics, manufacturing and purchasing have already changed dramatically to organize the decomposed functions over an integrated database. This will require organizations to undergo significant internal restructuring to breakdown internal functional barriers. Thus it is essential to automate and integrate the business functions at all levels of a production-logistics system by adapting an information technology over a common database. This is realized by advanced production planning and control techniques which also accelerate velocity of the flow of information and products across the industrial pipeline.

Distribution Resource Planning (DRP) and Manufacturing Resource Planning (MRP II) generate quick information flow across the entire industrial pipeline, where as Just-in-Time/Total Quality Management (JIT/TQM) generate quick material flow by driving responsive execution throughout the supply pipeline. Finally, Electronic Data Interchange (EDI) and bar coding enhance an efficient method to control information and material flow. Once the total channel integration is realized by each member of the marketing channel (manufacturer, wholesaler/distributors and retailers), customer connectivity links everything together.

Logistics is the provision of goods and services from a supply point to a demand point. A complete logistics system covers the entire process of moving raw materials and input requirements from suppliers to plants, the conversion of the inputs into products at certain plants, the movement of the products to various warehouses or depots and the eventual delivery of these products to the final customers. So logistics planning is a complex process consisting of multiple tasks and requiring effective coordination throughout the organization.

Today, JIT manufacturing practice requires integrating production, distribution and marketing & sales decision processes, so that inventory levels can be reduced along the whole production-distribution channel and customer demand can be satisfied on time. So, these decisions should be optimized concurrently as well as their corresponding organizational units in a firm should be integrated.

The integration of business applications over information technology infrastructure is just one aspect in world-class manufacturing management. Integrating production-logistics system also requires optimizing the inherent decision problems of all business functions simultaneously to avoid conflicts resulting from local optima. Managing production and logistics functions independently, with little or no coordination between them, works acceptably well if there is sufficient finished goods inventory to buffer the production and logistics operations from each other. However, the cost of carrying inventory and the trend to JIT operations create pressure to reduce inventories in the marketing channel. As a result of this pressure, many companies are exploring closer coordination along this channel. An integrated view of the entire production-logistics system may generate significant savings by trading off the costs associated with the whole, rather than minimizing production and logistics costs separately. Thus the integration of optimization practices along the supply chain poses a crucial issue to be solved.

Organizational structure is an important factor to support coordination of information flow in the supply chain management. In most companies, the organization structure is not designed to support the integration of business applications and optimization practices, and hence most efforts are directed at improvements within the existing designs. However, these efforts are reaching the point of diminishing returns so the time is right to consider making the necessary organizational changes that will facilitate coordination of these business functions and develop an ability to make more complex decisions within this organizational structure. Clearly the organizational designs should be made as a response to satisfy integration and optimization requirements.

As it is well-known, mathematical models provide a precise language and clear statement of the organizational structure and better understanding of the managerial

choices for the organizational design. Thus the first aim of this study is to describe the organizational structure of a production-logistics system within the framework of a single mathematical model from the decision processes and information systems point of view. This is in fact an attempt to describe the system with *centralized* decision-making.

However, centralized planning naturally leads to a complex, large-scale model which is difficult to solve optimally. Hence it becomes a necessity to develop alternative approaches of optimization techniques which provide near optimal solutions for all the parties involved in the integrated model. Decomposition techniques such as hierarchical decomposition, Lagrangean relaxation, Dantzig-Wolve (price-directive) and Benders (resource-directive) decomposition techniques are widely used as tools to solve such large-scale integrated models.

Consequently the aim of this study is converted into an attempt to develop mathematically sound and robust procedures as tools for establishing decentralized structures without too much sacrifice of global integration and optimization. An underlying assumption of this study is that organizational designs should be made corresponding to the structure of information flow in an efficient mathematical model. As it is well known, different decomposition and search methods imply different organizational designs. First a good solution procedure should be obtained and proven to be robust for a certain decomposition style, and only afterwards a manufacturing company should be organized in the manner dictated by the decomposition method. This study first proposes a certain hierarchical decomposition scheme and develops efficient and fast solution methods using information technology advances, and it is required that the decomposition style implied by this decomposition should be clearly implemented in the organizational design of a company.

This study decomposes the integrated model with centralized planning into two submodels. The first submodel tries to optimize the production and distribution functions simultaneously while the second model addresses the vehicle routing decisions based on the optimal production and distribution decisions. This hierarchical decomposition creates a *partially decentralized* organization. However in the process of solving the first-level

aggregate model by the Lagrangean relaxation method, it is realized that the model further decomposes into distribution and production subproblems naturally, and a *decentralized* organization arises in the management of the downstream supply chain.

The second level of the hierarchical decomposition which corresponds to the vehicle routing is a combinatorial optimization problem. Recently, five metaheuristic approaches have emerged for handling complex decision problems which are genetic algorithms, neural networks, simulated annealing, tabu search and target analysis. This study proposes a tabu search algorithm for the second level problem which performs pretty well compared to the other tabu search algorithms.

The study is organized as follows: Literature survey about the related subjects is presented in Section 2. Section 3 defines the production-logistics problem in this study. Section 4 provides a mathematical formulation of the integrated production-logistics model and explains the hierarchical structure proposed for the integrated production-logistics system. Lagrangean relaxation technique used in the solution of the aggregate production-distribution problem of the first level in the hierarchical decomposition, and solution procedures of the associated production and distribution subproblems are discussed in Section 5. Vehicle routing problem which corresponds to the second level in the hierarchical decomposition, and the proposed solution procedure are explained in Section 6. Section 7 gives the experimental design and computational results of the study. Finally, Section 8 states the conclusions and suggestions for future study.

2. LITERATURE SURVEY

The research areas which inspired this study can be classified into hierarchical planning, integrated manufacturing-distribution systems, the vehicle routing problem and metaheuristics.

Hierarchical planning:

Hierarchical production planning provides an encouraging approach to overcome the impracticality and the complexity in applying a monolithic approach to a large-scale problem. The basic idea of this procedure is to reduce problem complexity by decomposing the planning process into isolated subproblems, each of which may be solved using appropriate models.

Decomposition of the planning process is an important element in hierarchical planning, and the length of the decision horizon and level of the decisions are important aspects in the planning process. Hierarchical approach defines interdependencies in a great variety of ways. One approach is to define a hierarchical product structure as proposed by Hax and Meal [1] in terms of items, families and types and formulate the resulting decision problems accordingly. Another issue in defining subproblems is to consider the existing organizational structures adequately. In this respect, Schneeweiss [2] gives a general framework for hierarchical interdependencies within an organization. He presents a uniform conceptual framework to discuss the structure of hierarchical negotiations, principal-agent relationships, various kinds of hierarchical planning procedures, and hierarchical algorithms.

Dempster *et al.* [3] provides a first survey of fields of application of hierarchical planning, and show that this type of problems may also be modeled as a multistage stochastic program.

Simchi-Levi [4] proposes a hierarchical approach to the design and control of probabilistic distribution systems where only a subset of all potential customers needs service on any given working day. Three main subproblems corresponding to the three levels of decision - strategic, tactical and operational- are integrated into a hierarchical design, and the total system cost is proven to be asymptotically optimal.

In principal, hierarchical planning is based on a top-down control. However infeasibilities on the lower levels, or inconsistencies due to disaggregation may require some modifications. To solve these problems, Günther [5] suggests that three types of decision making on the superior level have to be considered: fixed, frozen and tentative plans by using the rolling horizon approach with anticipation.

Saad [6] extends the earlier hierarchical production planning approaches initiated by Hax and Meal into a hierarchical model for multi-plant, multi-product firms which incorporates both hierarchical and functional interface interacting simultaneously as in practice. He uses goal programming to solve the problem of feasibility which arises in disaggregating the aggregate production plan into detailed schedules at the product-family disaggregation level, and uses the branch and bound at the item-disaggregation level.

Kistner and Steven [7] reviews applications of operations research to solve planning problems which arise in subsystems of production control. They also consider some theoretical aspects of hierarchical production planning. They state that heuristic decomposition may be applied for tuning decisions on the tactical and operative levels of the planning process.

Graves [8] decomposes a large scale production planning problem in the context of Hax and Meal's hierarchical framework. He proposes a hybrid approach by applying Lagrangean relaxation technique to reduce differences between targets set by aggregate production planning and lot sizes for articles. A similar approach based on Benders' cuts is used by Aardal and Larsson [9].

In hierarchical planning, data and decision variables have to be aggregated to reduce complexity. Initially aggregation theory has been developed in the area of

macroeconomics. Axsäter [10], [11] and Axsäter *et al.* [12] recognize its importance for managerial planning on an aggregate level and applies its results to hierarchical production planning. As aggregation always involves loss of information, aggregation theory has to concentrate on infeasibilities caused by inevitable aggregation errors and their consequences on decisions and objectives. Erschler, Fontan and Merce [13] considers the consistency of aggregation in a rolling horizon environment. They show that the knapsack algorithm in connection with the look-ahead rule guarantees the consistency.

A study by Tsubone, Matsuura and Tsutsu [14] verifies the validity of a hierarchical production planning system for a two-stage process. They clarify, through a simulation model, the relationship between the production planning rules and the buffer inventory role in terms of manufacturing performance in order to support the choice of an optimum production system.

Integrated Manufacturing-Distribution Systems:

Most companies manage production and distribution functions independently with little or no coordination between production scheduling and distribution planning. This decoupled approach works acceptably well if there is sufficient finished goods inventory to buffer the production and distribution operations from each other. However the cost of carrying inventory and the trend to just-in-time operations is forcing to reduce inventories in the distribution channel. So companies need to coordinate production and distribution planning.

Glover *et al.* [15] develops a network flow model of the production scheduling, inventory and distribution decisions of Agrico Chemical. They embed this model in a decision support system that is used to analyze both short-run planning decisions and long-range strategic decisions.

Bloemhof-Ruwaard, Salomon and Wassenhove [16] proposes and analyses alternative model formulations for the problem of coordinating product and by-product flows in a two-level distribution network. Their problem is an extension of the classical facility location problem in which they consider a variant of the two-level location

problem. They analyze lower bounding procedures, based on linear programming and Lagrangean relaxation. Although the quality of the linear programming based lower bounds is on the average better than the quality of the Lagrangean lower bounds, the Lagrangean lower bounds are computed faster for larger sized problem instances. They also propose and compare a number of new upper bounding heuristics for their problem. The way they design their computational experiments is similar to the one prepared in this study.

Matta [17] deals with the problem of scheduling the production of several products on a single production line where product changeovers incur setup cost and/or time. This kind of problem has a trade-off between changeover and inventory carrying costs. This is a difficult problem since the number of possible schedules grows exponentially with the number of products and periods. He develops an iterative procedure that uses the Lagrangean technique to find lower bounds, and an efficient heuristic procedure to obtain feasible schedules from Lagrangean solutions. He eliminates the inventory variables in the model by substitution as it is done in this study.

Traditionally, the economic lot size problem of raw materials and finished products is treated separately which may result in suboptimization. Integrated optimization of these decision subsytems that complement each other are more preferable. This means subsystems should be linked together and corresponding decisions should be taken in a coordinated manner. Hence the degree of suboptimality is reduced. A study by Goyal and Deshmukh [18] reviews the literature on integrated procurement-production (IPP) systems by classifying IPP models into several categories.

Rangan and Jaikumar [19] shows that there is a close link between strategic decisions (i.e., the number of levels between the producer and the customer) and tactical decisions (i.e., channel management policies such as trade discounts and rebates) in the design of distribution systems by applying an integrated model that solves the strategic issue of channel levels and the tactical issue of price rebates simultaneously in C. C. Korns Company. The goal is to structure the optimal buying arrangement such that customers minimize their procurement costs simultaneously while the manufacturer maximizes profits.

Pyke and Cohen [20] develops a model of an integrated production and distribution system comprised of a single station model of a factory, a stockpile of finished goods (FG), and a single retailer. Multiple products with stochastic, independent demand are produced at the factory stored at FG which may also order an expedite batch of a particular product, and distributed to the retailer where demand is met or backlogged. They impose no capacity constraint on FG or the retailer while the factory is capacity constrained. They propose an algorithm which is proved to be near-optimal for limited tests.

Federgruen and Zheng [21] develops a model for a capacitated production / distribution network of general (but acyclic) topology with a general bill of materials, as considered in MRP or DRP systems which assumes stationary, deterministic demand rates and a standard stationary cost structure. There exist capacity constraints which consist of bounds on the frequency with which individual items can or need to be replenished. A pair of simple and efficient algorithms capable of determining an optimal power-of-two policy is derived by the researchers.

Soumis, Sauve and Beau [22] deals with the origin-destination assignment and the vehicle routing problem simultaneously. They present a two phase solution method for this problem using a minimum cost flow model followed by heuristic tour construction and improvement procedures. The first phase corresponds to strategic planning while the second phase corresponds to operational planning.

Both inventory allocation and vehicle routing decisions are closely interrelated logistical problems. Chien, Balakrishnan and Wong [23] addresses the problem of distributing a limited amount of inventory among customers using a fleet of vehicles so as to maximize profit. They formulate the integrated problem as a mixed integer program and develop a Lagrangean relaxation based solution procedure which exploits the underlying special structure of the problem to produce good upper bounds and heuristic solutions.

Chandra and Fisher [24] investigates the value of coordinating production and distribution planning by considering a plant that produces a number of products over time and maintains an inventory of finished goods at the plant. They compare two approaches,

one in which the production scheduling and vehicle routing problems are solved separately, and another in which they are coordinated within a single model.

Iyogun [25] describes an algorithm for a distribution problem involving several retailers and several warehouses and multiple items where external demands for items are constant, continuous and deterministic and each facility has a fixed setup cost and incurs echelon holding cost for any unused stock.

Iyogun and Atkins [26] presents an algorithm for obtaining a stationary product-nested policy for the multistage and multifacility pure distribution network where a facility at the end of the distribution network experiences a deterministic and continuous demand, and each facility has an echelon holding cost rate for each item it distributes and a facility dependent setup cost.

Axsäter [27] deals with an inventory system which consists of one warehouse and N retailers with constant lead times and Poisson demand. He provides simple recursive procedures for determining the holding and shortage costs of different control policies.

The vehicle routing problem:

The *Vehicle Routing Problem* (VRP) can be described as the problem of designing optimal delivery or collection of routes from one or several depots to a number of customers subject to side constraints. There exists a wide variety of VRPs and a broad literature on this class of problems. The variants of VRP include VRP with time windows, stochastic VRP, multi-depot VRP and others. There are a large variety of exact and approximate solutions developed for the VRP.

Laporte [28] provides an extensive overview of exact and approximate algorithms for VRP. He also summarizes some of the most representative or significant work published since 1985 on VRP and its variants in [29].

Exact algorithms for the VRP can be classified into three broad categories which are direct tree search methods, dynamic programming and integer linear programming. Some of the related studies are given below:

Laporte *et al.* [30] develops a branch-and-cut algorithm for the VRP with capacity and distance restrictions. The algorithm uses a two-index vehicle flow formulation including generalized subtour elimination constraints. Computational results for randomly generated instances up to 60 customers are presented.

Labbé *et al.* [31] proposes an exact branch-and bound algorithm for the capacitated VRP on trees. Lower bounds are based on the solution of bin packing problems and a heuristic with worst-case performance ratio of two is described.

Laporte, Mercure and Nobert [32] describe a branch-and bound algorithm for a class of asymmetrical VRP which solves instances up to 40 vertices. Fischetti, Toth and Vigo [33] presents a branch-and-bound algorithm which embeds two newer bounding procedures.

Fischer [34] proposes an exact branch-and-cut algorithm for the vehicle routing with capacities in which single customer trees are disallowed. The problem is modeled as that of determining a minimum cost K-tree with two edges incident to the depot. Lower bounds are obtained by dualizing the side constraints in a Lagrangean fashion.

Christofides, Mingozzi and Toth [35] studies the dynamic programming of VRP. They introduce a *state-space relaxation* procedure to reduce the number of states which provides a longer bound on the cost of the optimal solution. The optimum can then be reached by embedding the bounding procedure in an enumerative scheme.

Balinski and Quandt [36] is among the first to propose a set partitioning formulation for VRPs. To overcome the difficulties encountered with this formulation, the *column generation algorithm* is used which has been applied by Rao and Zionts [37], Foster and Ryan [38], Orloff [39], Desrosiers et al. [40], and Agarwal *et al.* [41].

Fischer and Jaikumar [42] has developed a three-index vehicle flow formulation for VRPs with capacity restrictions, time windows and no stopping times. They have also developed an algorithm based on this formulations. A two-index vehicle flow formulation in symmetrical capacitated VRPs, and time or distance constrained VRPs has been proposed by Laporte *et al.* [43].

As for the approximate algorithms, some of the many studies are given below:

Gillett and Miller [44] introduces and illustrates the *sweep algorithm* for solving the VRP with load and distance constraints for each vehicle. The locations that are used to make up each route are determined according to the polar-coordinate angle for each location. An iterative procedure is then used to improve the total distance traveled over all routes.

Paessens [45] sets criteria for the application of several heuristics (sweep method, two-phase method and savings method which has been first proposed by Clarke and Wright [46]) for solving the VRP. Since in practice, it is more comfortable to use microcomputers, he presents modifications of the savings method which takes less CPU time and reduced storage requirements, so that the savings method can be implemented on microcomputers.

Christofides and Eilon [47] considers three solution methods for the VRP which are *a branch-and-bound approach, the "savings" approach* and *the three-optimal tour method.* The excessive computation time and computer storage required for the first method renders it impracticable for large problems. They examine ten problems and the results suggest that the last method is superior to the other two methods.

Gaudioso and Paletta [48] deals with the optimal management of periodic deliveries of a given commodity. Their goal is to assign, over a planning period, a feasible combination of delivery days to each customer and to determine the scheduling and routing of the vehicles, minimizing the fleet size. They propose heuristic algorithms and report computational experience.

Dell'Amico, Fischetti and Toth [49] deals with the multiple depot vehicle scheduling problem in which the aim is to minimize the number of vehicles used and the overall operational cost. They design a new polynomial-time heuristic algorithm and propose several effective refining procedures.

Metaheuristics:

Recently, five approaches have emerged for handling complex decision problems: genetic algorithms, neural networks, simulated annealing, tabu search, and target analysis. The first two -genetic algorithms and neural networks- are inspired by principles derived from biological sciences, and simulated annealing derives from physical science, notably the second law of thermodynamics. Tabu search and target analysis stem from the general tenets of intelligent problem-solving.

Artificial intelligence is a revived approach to problem-solving that requires heuristic search intrinsically in knowledge-base operations, especially for logical and anological reasoning mechanisms. So, one bilateral linkage between operations research and artificial intelligence is their common interest in solving hard problems with heuristic search. Glover and Greenberg [50] considers the above five approaches of heuristic search for solutions to combinatorially complex problems in artificial intelligence.

Osman [51] reviews, classifies, analyzes and reports brief descriptions of the recently emerging heuristic approaches for combinatorial optimization problems. He first defines combinatorial optimization problems and their complexity. Since exact algorithms can only solve small-sized problems of this type, approximate algorithms are needed to provide near optimal solutions in reasonable amount of computation time. He focuses on recent approaches derived by analogical reasoning from artificial intelligence and natural science. He also describes the basic local search methods and shows how the metastrategy simulated annealing and the tabu search algorithms can guide and continue the search beyond the point of local optimality obtained by an embedded local search heuristic, so that improved solutions can be obtained.

Since the VRP is an NP-hard combinatorial optimization problem, most researchers have concentrated on the development of heuristics. Many of the latest developments have occurred in the area of metaheuristics. Gendreau, Laporte and Potvin [52] surveys recent applications of metaheuristics to the VRP.

Alfa *et al.* [53] applies the Simulated Annealing to the capacitated version of the VRP, while Osman [54] considers maximum route durations as well. The latter implementation is more involved in several respects.

The literature on genetic algorithms for VRPs is scarce. Gendreau, Laporte and Potvin [52] describes three known approaches to this problem. The work of Blanton and Wainwright to solve VRP with time windows is a hybridization of a genetic algorithm based on the GENITOR package with a greedy insertion heuristic. GIDEON is a genetic algorithm for VRPs with time windows, based on a "cluster-first, route-second" strategy. Finally, GENEROUS is the genetic routing system for VRPs with time windows, which eliminates the difficulties related to the encoding of a solution into a chromosome by applying the crossover and mutation operators on the solutions themselves.

Two early TS algorithms applied to VRP due to Willard, and Pureza and França are described in Gendreau, Laporte and Potvin [52]. Osman [54] also uses TS to solve VRP. In one version of his algorithm called BA (best-admissible), the whole neighbourhood is explored and the best non-tabu feasible move is selected; in the other version, FBA (first-best-admissible), the first admissible improving move is selected if one exists; otherwise the best admissible move is implemented;

TABUROUTE algorithm of Gendreau, Hertz and Laporte [55] is a new tabu search heuristic for the vehicle routing problem with capacity and route length restrictions. TABUROUTE allows infeasible solutions by adding penalty terms in the objective function if the solution is infeasible. It considers a sequence of adjacent solutions obtained by repeatedly removing a vertex from its current route and reinserting it into another route using GENI, a generalized insertion procedure developed by Gendreau, Hertz and Laporte [56] for the traveling salesman problem. The GENI procedure consists of inserting a vertex

between two of its p closest neighbors on a route while performing a local reoptimization of the route.

Taillard [57] presents two partition methods that speed up iterative search methods applied to VRPs including a large number of vehicles. He shows that it is possible to decompose VRP into subproblems that may be solved independently. He uses a simple implementation of TS as an iterative search method. Rather than executing the insertions with GENI, he uses standard insertions, thus enabling each insertion to be carried out in less time, and feasibility is always maintained. A neighborhood search procedure similar to the above methods in some aspects is developed in this study.

3. PROBLEM DEFINITION

Most producers do not sell their goods directly to the final users. They work with marketing intermediaries to bring their products to market. The marketing intermediaries make up a *marketing channel* (also called a *trade channel* or *distribution channel*). Intermediaries smooth the flow of goods and services, reduce the number of contacts and the amount of work that must be done, and bridge the discrepancy between the assortment of goods and services generated by the producer and the assortment demanded by the consumer.

Some intermediaries -such as wholesalers and retailers- buy, take title to, and resell the merchandise; they are called merchants. Others -such as brokers, manufacturers' representatives and sales agents- search for customers and may negotiate on the producer's behalf but do not take title to the goods; they are called agents. Still others -such as transportation companies, independent warehouses, banks and advertising agencies- assist in the distribution process but neither take title to goods nor negotiate purchases or sales; they are called facilitators.

A supply chain model includes the suppliers of the manufacturer into the same framework and aims to coordinate the activities of suppliers, manufacturers, wholesalers and retailers. This study addresses only the downstream supply chain activities, which is traditionally known as the distribution channel. A conventional distribution channel comprises an independent producer, wholesaler(s), and retailer(s). Each is a separate business entity seeking to maximize its own profits, even if this goal reduces profit for the system as a whole. No channel member has complete or substantial control over the other members. Recently new distribution systems have developed under the pressure of integration challenges. A vertical distribution system comprises the producer, wholesaler(s), and retailer(s) acting as an integrated system. One channel member owns the others or franchises them or has so much power that they all cooperate. The vertical

distribution system can be dominated by the producer, the wholesaler or the retailer. In horizontal distribution system, two or more unrelated companies put together resources or programs to exploit an emerging marketing opportunity. Each company lacks the capital, know-how, production or marketing resources to venture alone, or it is afraid of the risk. Multichannel distribution occurs when a single firm uses two or more distribution channels to reach one or more customer segments.

This study will analyze the production-logistics system of a manufacturing environment which is shown in Figure 3.1. The model consists of a plant which is faced with a capacitated lot-sizing problem in manufacturing a number of products over time. The products are distributed from the plant inventory to several wholesalers which are assumed to be independent from each other and each of which maintains an inventory of their own. Then the products are distributed from depots to a number of customers with deterministic time-varying demand requirements. The wholesalers may be owned by the sister company or may be operated as distinct companies. In the latter case the manufacturer is assumed to possess the freedom to ship any quantity to any wholesaler considering the customer demand requirements. Throughout the study the wholesalers will be called "depots". In this study, customers are not necessarily the end users. Although the plant and the depots possess the inventory holding capability, customers require JIT delivery of products, and backlogging is not permitted.

The aim of this study is to solve the above mentioned problem by optimizing the marketing & sales, production and logistics decisions concurrently. The following decisions are considered in the production-logistics system: The manufacturer should decide the production quantity, inventory level and shipment quantity to each depot for each of its product in each period. Each depot should decide upon its inventory level for each product in each period, and the quantity it should ship to each customer for each product so that customer demand is satisfied in all times and for all products. The depots should also decide upon the required number of vehicles and their corresponding routes so that overall production-logistics cost is minimized.

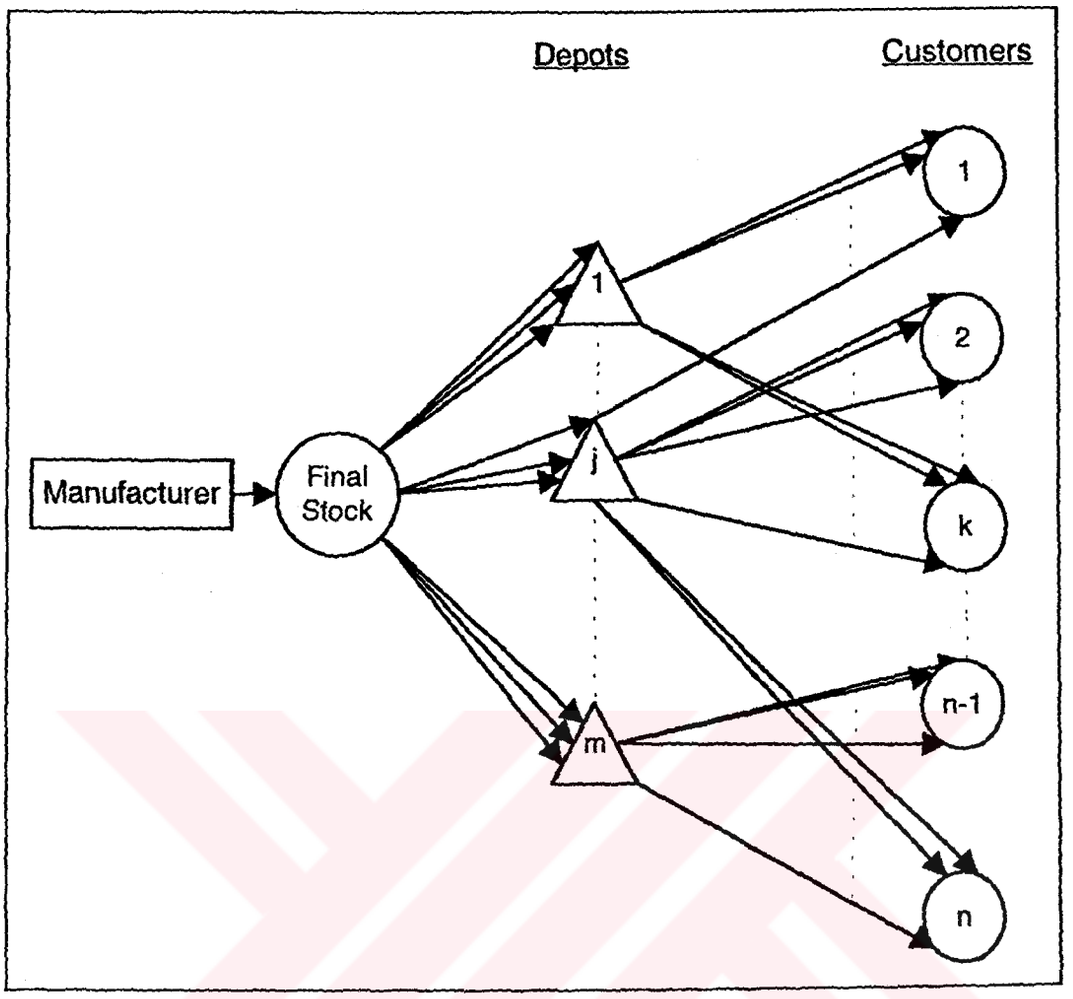

FIGURE 3.1. The production-logistics system of a manufacturing environment

In the hierarchical decomposition with partially decentralized planning the manufacturer is implicitly faced with an assignment problem of assigning customers to depots and coordinating the inventory and shipment decisions of all depots. However the Lagrangean relaxation further decomposes the aggregate model such that there is a central agent above all units which collects independent production and distribution subproblem solutions given by the manufacturer and the depots respectively, and updates and dictates new price data to these subunits. Figure 3.2 shows the information flow in the Lagrangean relaxation procedure.

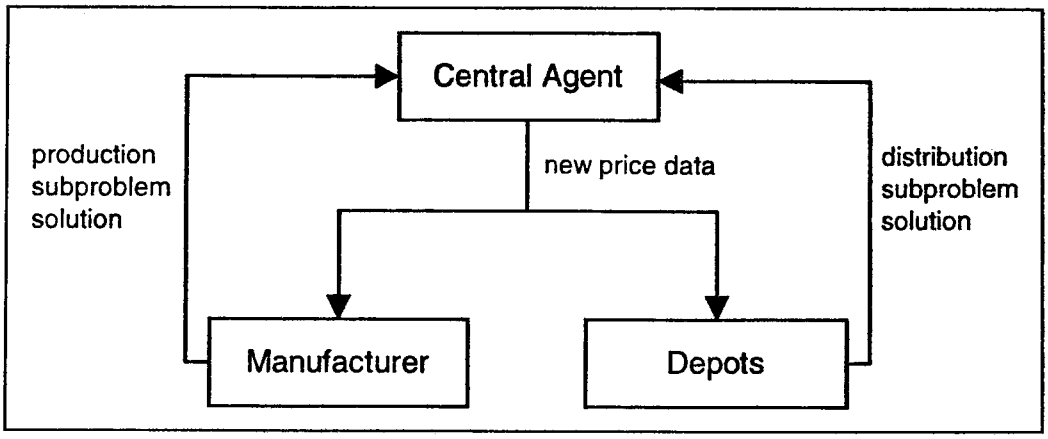

FIGURE 3.2. The Lagrangean relaxation information flow

Furthermore, in Lagrangean heuristic, the assignment decisions are first made by the depots in the distribution submodel. These decisions which are feasible with respect to manufacturer capacity are given to the manufacturer which solves the production problem to satisfy depot requirements. Both manufacturer and depots give their feasible decisions to the central agent. Figure 3.3 shows the information flow in the Lagrangean heuristic procedure.

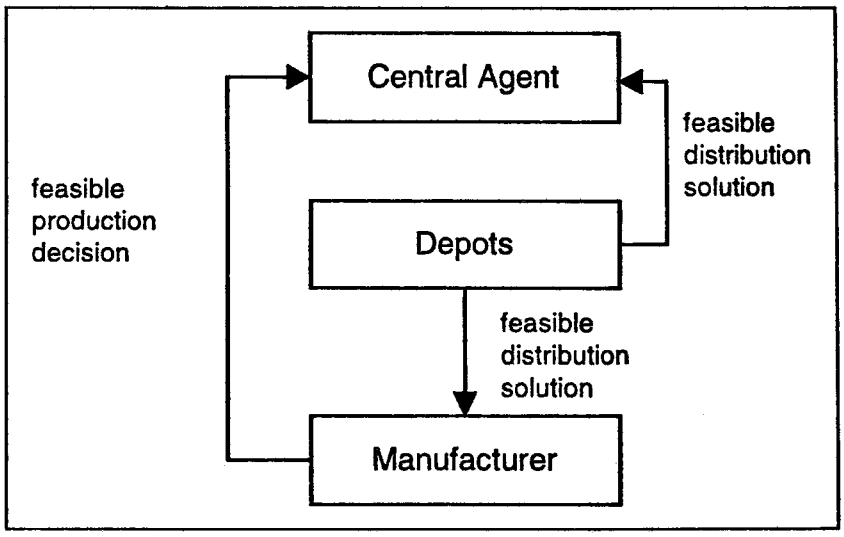

FIGURE 3.3. The Lagrangean heuristic information flow

4. HIERARCHICAL DESIGN OF THE INTEGRATED PRODUCTION-LOGISTICS SYSTEM

Integrating production-logistics system involves synchronized planning of manufacturing, distribution and vehicle routing decisions corresponding to the model illustrated in Figure 3.1.

As it is well known, mathematical models provide a precise language and clear statement of the organizational structure and better understanding of the managerial choices for the organizational design. Thus the first aim of this study is to describe the organizational structure of a production-logistics system within the framework of a single mathematical model from the decision processes and information systems point of view. This model is described in Section 4.1. This in fact an attempt to describe the system with centralized decision-making. However centralized planning leads to a complex, large scale model which is difficult to solve. Thus decomposition is needed to decentralize the decision process.

This study proposes to decompose the integrated model with centralized planning into two submodels: The first submodel tries to optimize the production and distribution functions simultaneously while the second submodel addresses the vehicle routing decisions based on the optimal production and distribution decisions. This hierarchical decomposition creates a partially decentralized organization. However in the process of solving the first-level aggregate model by the Lagrangean relaxation method, it is realized that the model has some internal structure which facilitates further decomposition, so that a decentralized organization arises. Decentralization is discussed in Section 4.2.

4.1. Integrated Production-Logistics Model

In this study, a mixed integer mathematical model is developed to address production, distribution and vehicle routing decisions in a typical manufacturing-distribution organization. This integrated model represents centralized planning since all the three problems are solved synchronizedly in a monolythic way. The objective of the model is to schedule all the decisions so as to minimize the total cost of production, distribution and vehicle routing.

Assumptions:

(a) The customer demand is known and time-variant. Backlogging of demand is not planned;

(b) Although setup time is assumed to be negligible, thus not included in the model, there exists considerable setup cost in manufacturing which is sequence-independent;

(c) There is a fixed cost and variable cost associated with transporting material from the manufacturer to the depots and from the depots to the customers. Variable costs are proportional to the transported amount of material;

(d) All the cost figures are time and item variant; furthermore fixed and variable costs of distribution depend both on the source and the destination;

(e) The manufacturer has a limited production capacity while there is no limitation upon stock holding capability of the depots;

(f) Distribution, manufacturing, and vehicle routing lead-times are negligible compared to the time-bucket, so they are not included in the formulation;

(g) The holding cost of customers is assumed to be very large, so that customer demand be satisfied exactly in the required period on a JIT basis.

Indices:

$i \in \{1...I\}$= set of product

$j \in \{1...m\}$= set of depots

$k \in \{1...n\}$= set of customers

$t \in \{1...T\}$= set of periods in the planning horizon

$v \in \{1...V\}$= set of vehicles

$p, q \in C_j=\{j, 1,...n\}$= set of probable nodes that can be visited by a vehicle leaving depot j

Decision Variables:

X_{it} = production amount of product i in period t

IO_{it} = ending inventory of product i in the plant in period t

YO_{ijt}^{k} = amount of product i transported from the plant to the depot j in period t intended for customer k

I_{ijt}^{k} = amount of product i kept in the stock of depot k in period t

Y_{ijt}^{k} = amount of product i shipped from depot j to customer k in period t

Q_{pq}^{ijktv} = quantity of product i shipped in vehicle v from depot j at time t destined for customer k over the arc (p,q)

$$Z_{it} = \begin{cases} 1 & \text{if } X_{it} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$WO_{ijt}^{k} = \begin{cases} 1 & \text{if } YO_{ijt}^{k} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$W_{ijt}^{k} = \begin{cases} 1 & \text{if } Y_{ijt}^{k} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$R_{pq}^{ijktv} = \begin{cases} 1 & \text{if a vehicle v leaving depot j at time t transports product i for customer k} \\ & \text{traverses arc (p,q)} \\ 0 & \text{otherwise} \end{cases}$$

Parameters:

s_{it} = setup cost of producing product i in period t

p_{it} = unit variable cost of producing product i in period t

ho_{it} = unit holding cost of product i in the plant in period t

fo_{ijt} = fixed cost of transporting product i from plant to depot j in period t

co_{ijt} = unit variable cost of transporting product i from the plant to depot j in period t

h_{ijt} = unit holding cost of product i in depot j in period t

f_{ijt}^{k} = fixed cost of transporting product i from depot j to customer k in period t

dis_{pq} = distance between vertices p and q

a_i = processing time of product i (in hours)

A_t = total available production capacity in period t (in hours)

B_v = capacity of vehicle v

α =a scalar used to convert distance data to cost data

The integrated multi-item 2-stage production-logistics problem (PL) can be formulated as a mixed integer linear model:

$$Z(PL) = \min\Bigg[\sum_{t=1}^{T}\sum_{i=1}^{I} s_{it}Z_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} p_{it}X_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} ho_{it}IO_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} fo_{ijt}WO_{ijt}^{k}$$

$$+ \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} co_{ijt}YO_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} h_{ijt}I_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} f_{ijt}^{k}W_{ijt}^{k}$$

$$+ \alpha\sum_{t=1}^{T}\sum_{v=1}^{V}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{p\in C_j}\sum_{q\in C_j} dis_{pq}R_{pq}^{ijktv}\Bigg] \qquad (4.1.1)$$

subject to

$$\sum_{i=1}^{I} a_i X_{it} \le A_t \qquad \forall t \qquad (4.1.2)$$

$$X_{it} \le \left(\sum_{t=1}^{T} A_t \right) Z_{it} \qquad \forall i,t \tag{4.1.3}$$

$$IO_{i,t-1} + X_{it} - \sum_{j=1}^{m} \sum_{k=1}^{n} YO_{ijt}^{k} - IO_{it} = 0 \qquad \forall i,t \tag{4.1.4}$$

$$YO_{ijt}^{k} \le \left(\sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{k=1}^{n} d_{it}^{k} \right) WO_{ijt}^{k} \qquad \forall i,j,k,t \tag{4.1.5}$$

$$I_{ij,t-1}^{k} + YO_{ijt}^{k} - Y_{ijt}^{k} - I_{ijt}^{k} = 0 \qquad \forall i,j,k,t \tag{4.1.6}$$

$$Y_{ijt}^{k} \le \left(\sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{k=1}^{n} d_{it}^{k} \right) W_{ijt}^{k} \qquad \forall i,j,k,t \tag{4.1.7}$$

$$\sum_{j=1}^{m} Y_{ijt}^{k} = d_{it}^{k} \qquad \forall i,k,t \tag{4.1.8}$$

$$\sum_{p \in C_j} Q_{pr}^{ijktv} - \sum_{q \in C_{jj}} Q_{rq}^{ijktv} = 0 \qquad \forall j,t,i,k,v,r \neq k \tag{4.1.9}$$

$$\sum_{v=1}^{V} \sum_{p \in C_j} Q_{pk}^{ijktv} = Y_{ijt}^{k} \qquad \forall j,t,i,k \tag{4.1.10}$$

$$\sum_{i=1}^{l} \sum_{k=1}^{n} Q_{pq}^{ijktv} \le B_v \qquad \forall j,t,v,p,q,p \ne q \tag{4.1.11}$$

$$Q_{pq}^{ijktv} \le Q_v R_{pq}^{ijktv} \qquad \forall j,t,i,k,v,p,q,p \ne q \tag{4.1.12}$$

$$\sum_{p \in C_j} \sum_{v=1}^{V} R_{pq}^{ijktv} \le 1 \qquad \forall j,t,i,k,q \tag{4.1.13}$$

$$\sum_{q \in C_j} \sum_{v=1}^{V} R_{pq}^{ijktv} \le 1 \qquad \forall j,t,i,k,p \tag{4.1.14}$$

$$\sum_{p \in C_j} R_{pr}^{ijktv} - \sum_{q \in C_j} R_{rq}^{ijktv} = 0 \qquad \forall j,t,i,k,v,r \tag{4.1.15}$$

$$R_{pq}^{ijktv} \in S \qquad \forall i,j,k,t,v,p,q \tag{4.1.16}$$

$$Z_{it} \in \{0,1\}, \quad X_{it}, IO_{it} \ge 0 \qquad \forall i,t \tag{4.1.17}$$

$$WO_{ijt}^{k}, W_{ijt}^{k} \in \{0,1\}, \quad I_{ijt}^{k}, YO_{ijt}^{k}, Y_{ijt}^{k} \geq 0 \qquad \forall i,j,k,t \qquad (4.1.18)$$

$$R_{pq}^{ijktv} \in \{0,1\}, \quad Q_{pq}^{ijktv} \geq 0 \qquad \forall i,j,k,t,v,p,q \qquad (4.1.19)$$

The objective function (4.1.1) aims to minimize the total fixed and variable costs associated with production, inventory holding and 2-stage distribution. Capacity restriction in the plant is expressed by (4.1.2). The technical constraints (4.1.3) guarantee that setup be done prior to production. Constraints (4.1.4) express the material balance in the plant while constraints (4.1.6) on the other hand express the material balance in each depot. The technical constraints (4.1.5) and (4.1.7) ensure that distribution from the plant to the depots and distribution from depots to customers both generate fixed costs, respectively. Constraints (4.1.8) state that customer demand in period t be met exactly in period t on a JIT basis. With constraints (4.1.9) accumulation of product i at the vertice r destined for another customer is avoided. Constraints (4.1.10) assure that the planned quantity is sent to customer k. Capacity restriction of any vehicle is expressed by (4.1.11). Constraints (4.1.12) are technical constraints to ensure that if there exists a shipment using an arc by a vehicle, then this creates a cost to the objective function. Constraints (4.1.13) and (4.1.14) ensure that at most one vehicle is assigned to carry product i to customer k from depot j in period t. It is guaranteed in (4.1.15) that a vehicle leaves the demand vertex it has already entered. The subtour elimination constraints are defined in (4.1.16) where S can be defined in anyone of the ways found very frequently in literature.

The integrated production-logistics problem is a combinatorial optimization problem which is large in terms of variables and constraints even with a reasonable number of products, depots, customers and time periods. This model can not be solved numerically with reasonable efforts of time and computer capacity. Hence alternative approaches should be required to be developed to solve the problem.

4.2. Hierarchical Decomposition of the Production-Logistics System

Hierarchical decomposition partitions a problem into a hierarchy of subproblems. In any planning period, the subproblems are solved sequentially with the solutions of subproblems from the upper hierarchy imposing constraints on the lower hierarchy subproblems. Hence, the approach considers interdependencies appropriately and at the same time reduces problem complexity. In the design of an hierarchical planning system, the partitioning of the planning process, the selection of adequate submodels at each level and their coordination are the most essential aspects.

In this study, the integrated production-logistics system represented by (PL) is decomposed into two levels within the framework of hierarchical decomposition. The first level deals with production-distribution system while the second level deals with the vehicle routing problem. In the first level, aggregate production-distribution problem is solved to obtain the related decisions, and in the second level vehicle routing decisions will be addressed based on the top level decisions. Hierarchical decomposition of the whole system is given in Figure 4.2.1.

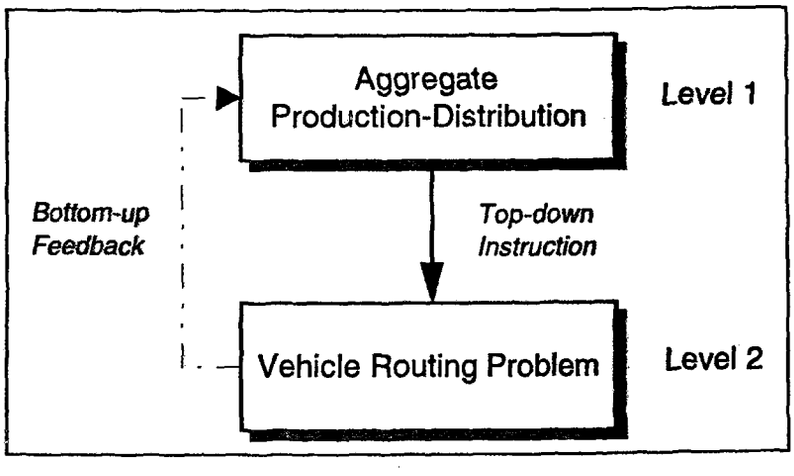

FIGURE 4.2.1. Hierarchical decomposition of the production-logistics system

Since hierarchical decomposition partitions the problem into two levels which are solved separately, it ends up with partially decentralized planning. The first level considers solving production and distribution problems simultaneously while incorporating some cost information from the lower level into its solution procedure.

4.2.1. Aggregate Production-Distribution Model

The aggregate production-distribution model in this study involves the related parts of the integrated production-logistics model plus some information from the vehicle routing problem. All the related assumptions, index set, parameters and decision variables given in section 4.1 are also valid here with the following parameter included:

c_{ijt}^{k} =unit variable cost of transporting product i from depot j to customer k in period t. It is computed by multiplying dis_{pq} with the scalar α.

The aggregate production-distribution (PD) can be formulated as a mixed integer model by taking the related parts of (PL) described in section 4.1 with the following modifications:

$$Z(PD) = \min \Bigg[\sum_{t=1}^{T} \sum_{i=1}^{I} s_{it} Z_{it} + \sum_{t=1}^{T} \sum_{i=1}^{I} p_{it} X_{it} + \sum_{t=1}^{T} \sum_{i=1}^{I} ho_{it} IO_{it} + \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{m} \sum_{k=1}^{n} fo_{ijt} WO_{ijt}^{k}$$

$$+ \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{m} \sum_{k=1}^{n} co_{ijt} YO_{ijt}^{k} + \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{m} \sum_{k=1}^{n} h_{ijt} I_{ijt}^{k} + \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{m} \sum_{k=1}^{n} f_{ijt}^{k} W_{ijt}^{k}$$

$$+ \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{m} \sum_{k=1}^{n} c_{ijt}^{k} Y_{ijt}^{k} \Bigg] \qquad (4.2.1)$$

subject to constraints (4.1.2)-(4.1.8),(4.1.17) and (4.1.18).

To eliminate $\{IO_{it}\}$ variables in (PD), constraints (4) can be written as:

$$IO_{it} = IO_{i0} + \sum_{r=1}^{t} X_{ir} - \sum_{r=1}^{t}\sum_{j=1}^{m}\sum_{k=1}^{n} YO_{ijr}^{k} \qquad \forall i,t \tag{4.2.2}$$

Substituting this expression into (PD), the following formulation (PD') is obtained.

$$Z(PD') = \min\Bigg[\sum_{t=1}^{T}\sum_{i=1}^{I} s_{it} Z_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I}\Big(p_{it} + \sum_{r=t}^{T} ho_{ir}\Big) X_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} fo_{ijt} WO_{ijt}^{k}$$

$$+ \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n}\Big(co_{ijt} - \sum_{r=t}^{T} ho_{ir}\Big) YO_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} h_{ijt} I_{ijt}^{k}$$

$$+ \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} f_{ijt}^{k} W_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} c_{ijt}^{k} Y_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I} ho_{it} IO_{i0}\Bigg] \tag{4.2.3}$$

subject to

$$\sum_{i=1}^{I} a_i X_{it} \le A_t \qquad \forall t \tag{4.2.4}$$

$$X_{it} \le \left(\sum_{t=1}^{T} A_t\right) Z_{it} \qquad \forall i,t \tag{4.2.5}$$

$$\sum_{r=1}^{t} X_{ir} \geq \sum_{r=1}^{t} \sum_{j=1}^{m} \sum_{k=1}^{n} YO_{ijr}^{k} - IO_{i0} \qquad \forall i,t \qquad (4.2.6)$$

$$YO_{ijt}^{k} \leq \left(\sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{k=1}^{n} d_{it}^{k} \right) WO_{ijt}^{k} \qquad \forall i,j,k,t \qquad (4.2.7)$$

$$I_{ij,t-1}^{k} + YO_{ijt}^{k} - Y_{ijt}^{k} - I_{ijt}^{k} = 0 \qquad \forall i,j,k,t \qquad (4.2.8)$$

$$Y_{ijt}^{k} \leq \left(\sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{k=1}^{n} d_{it}^{k} \right) W_{ijt}^{k} \qquad \forall i,j,k,t \qquad (4.2.9)$$

$$\sum_{j=1}^{m} Y_{ijt}^{k} = d_{it}^{k} \qquad \forall i,k,t \qquad (4.2.10)$$

$$Z_{it} \in \{0,1\}, \quad X_{it} \geq 0 \qquad \forall i,t \qquad (4.2.11)$$

$$WO_{ijt}^{k}, W_{ijt}^{k} \in \{0,1\}, \quad I_{ijt}^{k}, YO_{ijt}^{k}, Y_{ijt}^{k} \geq 0 \qquad \forall i,j,k,t \qquad (4.2.12)$$

For a typical production-distribution model with three products, four depots, six customers and 12 time periods, the problem (PD') consists of 1764 binary variables, 2628 real variables and 2892 constraints. Certainly this problem is again too large to be solved by anyone of the available software. So the Lagrangean relaxation is implemented to solve (PD') and the problem naturally decomposes itself into two subproblems, namely production and distribution subproblems. Here decoupled production and distribution decisions exchange information via the Lagrange multipliers. In Dantzig-Wolve (price-directive) and Benders (resource-directive) decomposition techniques, there is always a central agent (master problem) transmitting information between the subunits. In case of Lagrangean relaxation, there also exists a central agent in the form of generating Lagrange multipliers using the results of the decoupled subunits and dictating them to the production and distribution subunits. In every iteration, the central agent gathers information between production and distribution subunits about the current degree of inconsistency between them and revises its decision by updating Lagrange multipliers and finally dictates the information transmitted by the value of Lagrange multipliers to the two subunits for the next iteration. The technical details are provided in Section 5.

4.2.2. Vehicle Routing Problem

After solving aggregate production-distribution problem in the first level, disaggregation occurs in the second level based on the upper level decisions. In other words, the VRP is solved based on Y^k_{ijt} values obtained in the first level.

Since customer-product pairs (i,k) are already assigned to the depots in each period which are denoted by Y^k_{ijt} and W^k_{ijt}, a decentralized logistics management suffices to solve a single-depot VRP for each depot in each period. So for each depot j in any period t, corresponding customer vertices (i,k) are formed such that $Y^k_{ijt}>0$ for all i and k.

The information exchange between the VRP and the upper level decisions is done in the form of "top-down" and "bottom-up" manner as given in Schneeweiss [2]. the VRP and proposed solution procedures to solve it are discussed in details in Section 6.

4.3. Organizational Design

In hierarchical planning, either existing organizational structures may be used to define subproblems or business functions in the hierarchical decomposition are matched to the organizational units in a company. In this way, interdependencies and relations between the subproblems in the hierarchical structure and also between the organizations are defined appropriately.

Another aim of this study is to develop mathematically sound and robust procedures as tools for establishing decentralized structures. In fact an underlying assumption of this study is that organizational designs should be made corresponding to the structure of information flow in an efficient mathematical model. As it is well known, different decomposition and search methods imply different organizational designs. first a good solution procedure should be obtained and proven to be robust, for a certain decomposition style, and only afterwards a manufacturing company should be organized in the manner dictated by the decomposition method.

This study proposes a certain hierarchical decomposition scheme and develops efficient and fast solution methods using information technology advances. Thus the decomposition style can be clearly implemented in the organizational design of a company.

The implications of hierarchical decomposition proposed in this study corresponds to a decentralized organizational structure in a company. The first level in the decomposition which is the aggregate production-distribution problem naturally decomposes into distribution and production planning by applying Lagrangean relaxation technique although there is a high information exchange between them through the use of the Lagrange multipliers. Hence there becomes totally three levels of business functions in the whole system which have interactions and feedback among them. The first and second levels correspond to marketing & sales and production planning units in a company respectively. Since JIT philosophy exists over the whole system, distribution decisions pull manufacturing decisions, but when infeasibility occurs, this is provided as a natural

feedback from the production planning unit to the marketing & sales unit and the overall optimality is tried to be reached interactively. The third level which is the vehicle routing problem corresponds to the logistics unit in a company. The vehicle routing problem is considered in the first and second levels as an input to the cost function of the aggregate production-distribution problem. So there is a hidden feedback from the logistics unit to the upper units. Marketing & sales and production planning give their integrated decisions to the logistics unit as an input to its own problem. Feedback from the logistics unit to the upper level units is provided when necessary. By this way conflicts among critical departments in a company are tried to be minimized with the feedbacks provided in the hierarchical structure. Parallelism between business functions and organizational units is given in Figure 4.3.1

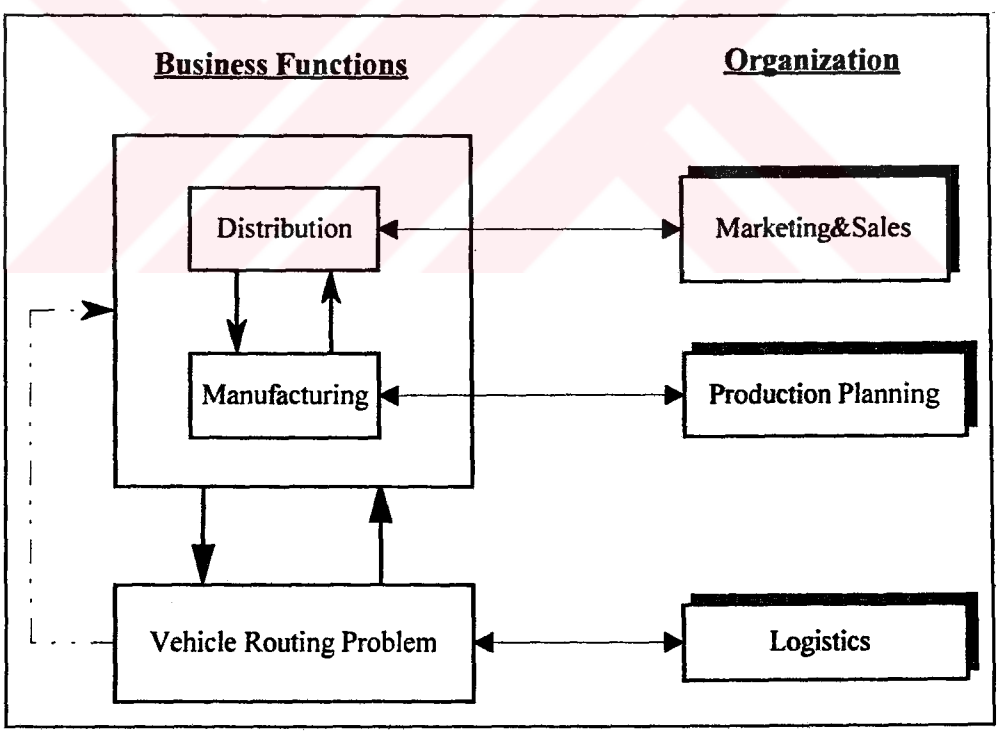

FIGURE 4.3.1. Parallelism between business functions and organizational units

5. LAGRANGEAN RELAXATION APPROACH TO THE AGGREGATE PRODUCTION AND DISTRIBUTION PROBLEM

As it is mentioned before, the mixed integer mathematical model given by (4.2.3) - (4.2.12) is developed to address production and two-echelon distribution decisions simultaneously. Since the problem is an NP-hard combinatorial optimization problem, it is often hard to solve it optimally in polynomial time for real-life problems which are by nature large in terms of variables and constraints.

As described in [58], one of the most computationally useful ideas of 1970s is the observation that many hard integer programming problems can be viewed as easy problems complicated by a relatively small set of side constraints. Dualizing the side constraints produces a Lagrangean problem that is easy to solve and whose optimal value is a lower bound for minimization problems on the optimal value of the original problem. So, Lagrangean relaxation approach is used to decouple the imbedded distribution and production decision subproblems and the subgradient optimization method is implemented to coordinate the information flow between these subproblems in a hierarchical manner.

In this section, first the construction of the Lagrangean relaxation technique will be described. Then, solution procedures for the two subproblems, the Lagrangean heuristic and computational results will be explained in detail.

5.1. Construction of the Lagrangean Relaxation Technique

The following is a combinatorial optimization problem formulated as the integer program given in [58].

$$
\begin{aligned}
&\textbf{(P)} && \min cx \\
&\text{subject to} && Ax = b \\
& && Dx \leq e \qquad (5.1.1) \\
& && x \geq 0 \text{ and integral.}
\end{aligned}
$$

Here x is n x 1, b is m x 1, e is k x 1, and all other matrices have conformable dimensions. It is assumed that the constraints of (P) have been partitioned into the two sets $Ax = b$ and $Dx \leq e$ so as to make it relatively easy to solve the Lagrangean problem

$$
\begin{aligned}
&\textbf{(LR}_u\textbf{)} && Z_D(u) = \min\ cx + u\,(Ax-b) \\
&\text{subject to} && Dx \leq e \qquad (5.1.2) \\
& && x \geq 0 \text{ and integral,}
\end{aligned}
$$

where $u = (u_1,\ldots,u_m)$ is a vector of Lagrange multipliers.

It is well known that $Z_D(u) \leq Z$. By using this fact, (LR_u) can be used to provide lower bounds for (P). Furthermore, good feasible solutions to (P) can frequently be obtained by perturbing nearly feasible solutions to (LR_u).

It is an important issue to determine u in Lagrangean relaxation applications. The best choice for u would be an optimal solution to the dual problem (D):

$$
\textbf{(D)} \qquad Z_D = \max_{u} Z_D(u) \qquad (5.1.3)
$$

The function $Z_D(u)$ has all the nice properties, like continuity and concavity, except differentiability. Although it is differentiable almost everywhere, it generally is nondifferentiable at an optimum point. Three most popular approaches for nondifferantiable optimization in Lagrangean relaxation applications are the subgradient method, two versions of the simplex method implemented using column generation techniques and multiplier adjustment methods. Among these, the subgradient method is easy to program and has worked well on many practical problems. So, in this study the subgradient method where gradients are replaced by subgradients is used to determine u values. Given an initial value u^0 a sequence $\{u^k\}$ is generated by the rule

$$u^{k+1} = u^k + \alpha^k (Ax^k - b) \tag{5.1.4}$$

where x^k is an optimum solution to (LR_u^k), α^k is a positive scalar step size and $(Ax^k - b)$ which corresponds to the dualized constraint set is the subgradient. The step size used most commonly in practice is

$$\alpha^k = \frac{\lambda^k \left(Z^* - Z_D(u^k)\right)}{\left\|Ax^k - b\right\|^2} \tag{5.1.5}$$

where λ^k is a scalar satisfying $0 < \lambda^k \leq 2$ and Z^* is an upper bound on Z_D, frequently obtained by applying a heuristic to (P).

5.2. Relaxed Production and Distribution Problem

Selecting between different constraint sets to be relaxed is an important issue in the Lagrangean relaxation method, because each constraint set requires different lengths of time for the solution of the problem with varying sharpness of bounds. Usually selecting a relaxation involves a tradeoff between these two properties; sharper bounds require more time to compute. In this study constraint (4.2.6) is chosen to be relaxed since it links the production and distribution problems. After arranging the terms in the objective function, the Lagrangean problem becomes as follows:

$$
\begin{aligned}
Z(L) = \min \Bigg[& \sum_{t=1}^{T}\sum_{i=1}^{I} s_{it} Z_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} (p_{it} + \sum_{r=t}^{T} ho_{ir} - \sum_{r=t}^{T} u_{ir}) X_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} fo_{ijt} WO_{ijt}^{k} \\
& + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} (co_{ijt} + \sum_{r=t}^{T} u_{ir} - \sum_{r=t}^{T} ho_{ir}) YO_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} h_{ijt} I_{ijt}^{k} \\
& + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} f_{ijt}^{k} W_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} c_{ijt}^{k} Y_{ijt}^{k} + IO_{it} \Big(\sum_{r=t}^{T}\sum_{i=1}^{I} ho_{ir} - \sum_{r=t}^{T}\sum_{i=1}^{I} u_{ir} \Big) \Bigg]
\end{aligned}
\qquad (5.2.1)
$$

subject to constraints (4.2.4), (4.2.5), (4.2.7)- (4.2.12).

Here $\{u_{it}: u_{it} \geq 0\}$ is the set of Lagrangean multipliers corresponding to (4.2.6), and the last two terms in the objective function can be disregarded in optimizing (L). Since this relaxation decouples the link between production and distribution decisions, it is easy to show that (L) decomposes into the subproblems (L1) and (L2) as given below:

Production Submodel L1:

$$Z(L1) = \min\left[\sum_{t=1}^{T}\sum_{i=1}^{I} s_{it} Z_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I}(p_{it} + \sum_{r=t}^{T} ho_{ir} - \sum_{r=t}^{T} u_{ir}) X_{it}\right] \qquad (5.2.2)$$

subject to constraints (4.2.4), (4.2.5) and (4.2.11).

Distribution Submodel L2:

$$Z(L2) = \min\left[\sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} fo_{ijt} WO_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n}(co_{ijt} + \sum_{r=t}^{T} u_{ir} - \sum_{r=t}^{T} ho_{ir}) YO_{ijt}^{k}\right.$$

$$\left. + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} h_{ijt} I_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} f_{ijt}^{k} W_{ijt}^{k} + \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{m}\sum_{k=1}^{n} c_{ijt}^{k} Y_{ijt}^{k}\right] \qquad (5.2.3)$$

subject to constraints (4.2.7)-(4.2.10) and (4.2.12).

5.2.1. Solution Procedure for Production Submodel

As it can be seen easily, the production submodel (L1) is further decomposable into T knapsack problems, each of which attempts to determine the capacitated production decision in each period. Thus the problem becomes solving T independent knapsack problems ($L1_t$) for t=1...T given by:

$$Z(L1_t) = \min\left[\sum_{i=1}^{I} s_{it} Z_{it} + \sum_{i=1}^{I} (p_{it} + \sum_{r=t}^{T} ho_{ir} - \sum_{r=t}^{T} u_{ir}) X_{it}\right] \qquad (5.2.1.1)$$

subject to constraints (4.2.4), (4.2.5) and (4.2.11) .

$L1_t$ can be optimally solved by using the following simple production rule. The proof is given below.

Production Rule (PR):

$$i^* = \arg\min\left\{\frac{p_{it} + \sum_{r=t}^{T} ho_{ir} - \sum_{r=t}^{T} u_{ir}}{a_i} + \frac{s_{it}}{A_t} < 0 \quad / i \in I\right\} \qquad (5.2.1.2)$$

$$X_{it} = \begin{cases} \dfrac{A_t}{a_i} & \text{for } i = i^* \\ 0 & \text{otherwise} \end{cases} \qquad (5.2.1.3)$$

where argmin$\{b_i \backslash\, i \in I\}$ is defined as the argument of the smallest b_i for all $i \in I$.

In any period, at most one item satisfying (5.2.1.2) is to be produced such that it uses all capacity. Production quantities of other items are set to zero. So $Z(L1_t)$ is either negative in which case production occurs in that period, or zero if there exists no item satisfying the condition given by (5.2.1.2).

Theorem: Production rule (PR) given by (5.2.1.2) and (5.2.1.3) solves the production problem ($L1_t$) optimally.

Proof: Let A_{it} be the amount of production capacity dedicated to produce product i in period t. So $\sum_{i=1}^{I} A_{it} = A_t$. Let S be a production schedule such that all products are produced with some amount and let v_{it} be the variable cost coefficient in production problem ($L1_t$).

$$v_{it} = p_{it} + \sum_{r=t}^{T} ho_{ir} - \sum_{r=t}^{T} u_{ir} \qquad (5.2.1.4)$$

Then there exists i and j such that

$$\frac{v_{it}}{a_i} + \frac{s_{it}}{A_t} < \frac{v_{jt}}{a_j} + \frac{s_{jt}}{A_t} \qquad (5.2.1.5)$$

Let S' be the production schedule obtained by dedicating A_j of product j to product i and setting the production quantity of j to zero. It is required to show that total production cost of production schedule S which is denoted by C(S) is bigger than the total cost of production schedule S' which is denoted by C(S').

$$C(S) = s_{1t} + v_{1t}\frac{A_{1t}}{a_1} + \ldots + s_{it} + v_{it}\frac{A_{it}}{a_i} + s_{jt} + v_{jt}\frac{A_{jt}}{a_j} + \ldots + s_{It} + v_{It}\frac{A_{It}}{a_I} \qquad (5.2.1.6)$$

$$C(S') = s_{1t} + v_{1t}\frac{A_{1t}}{a_1} + \ldots\ldots + s_{it} + v_{it}\frac{(A_{it} + A_{jt})}{a_i} + \ldots\ldots + s_{1t} + v_{1t}\frac{A_{1t}}{a_1} \qquad (5.2.1.7)$$

Let ΔC be the cost difference between C(S) and C(S').

$$\Delta C = C(S) - C(S') = A_{jt}\frac{v_{jt}}{a_j} - A_{jt}\frac{v_{it}}{a_i} + s_{jt} \qquad (5.2.1.8)$$

By (5.2.1.5) it is known that

$$\frac{v_{jt}}{a_j} - \frac{v_{it}}{a_i} + \frac{s_{jt}}{A_t} - \frac{s_{it}}{A_t} > 0 \qquad (5.2.1.9)$$

Multiplying both sides by A_{jt} gives the following expression:

$$A_{jt}\frac{v_{jt}}{a_j} - A_{jt}\frac{v_{it}}{a_i} + A_{jt}\frac{s_{jt}}{A_t} - A_{jt}\frac{s_{it}}{A_t} > 0 \qquad (5.2.1.10)$$

Hence it is required to show that

$$s_{jt} > A_{jt}\frac{s_{jt}}{A_t} - A_{jt}\frac{s_{it}}{A_t} \qquad (5.2.1.11)$$

Since $A_{jt}<A_t$

$$s_{jt} > A_{jt}\frac{s_{jt}}{A_t} > A_{jt}\frac{s_{jt}}{A_t} - A_{jt}\frac{s_{it}}{A_t} \qquad (5.2.1.12)$$

By (5.2.1.12), (5.2.1.11) is shown. Thus S' has less cost than S and is also possible to reduce the cost of the new schedule S' further if there exists i and j satisfying (5.2.1.5). This improvement continues until there exists one product in the production schedule. If producing this product leads to negative production cost, then it is decided to produce this item. If its cost is positive, then it is decided not to produce any product in period t.

Interpretation of Production Rule

The only expression to make the whole expression in (5.2.1.2) is $\sum_{r=t}^{T} u_{ir}$. The bigger the u_{ir} values, the smaller are the v_{it} values. This is reasonable since u_{ir} is related with cumulative shortage of product i until period t. Hence the production problem has information from the distribution problem in the values of u_{ir}. If the sum of cumulative shortage from period t to T is large enough to make v_{it} negative, the distribution subproblem signals "production" of that product to the production subproblem within the values of u_{ir}.

5.2.2. Solution Procedure for Distribution Problem

The distribution submodel (L2) is also decomposable into (i.k) subproblems, each of which represents the distribution problem of (i,k)th item-customer pair and for i=1...I, k=1...n is given by

$$Z(L2_{ik}) = \min \left[\sum_{t=1}^{T} \sum_{j=1}^{m} fo_{ijt} WO_{ijt}^{k} + \sum_{t=1}^{T} \sum_{j=1}^{m} (co_{ijt} + \sum_{r=t}^{T} u_{ir} - \sum_{r=t}^{T} ho_{ir}) YO_{ijt}^{k} \right.$$

$$\left. + \sum_{t=1}^{T} \sum_{j=1}^{m} h_{ijt} I_{ijt}^{k} + \sum_{t=1}^{T} \sum_{j=1}^{m} f_{ijt}^{k} W_{ijt}^{k} + \sum_{t=1}^{T} \sum_{j=1}^{m} c_{ijt}^{k} Y_{ijt}^{k} \right] \qquad (5.2.2.1)$$

subject to constraints (4.2.7)-(4.2.10) and (4.2.12).

For each (i,k) pair, a forward algorithm (DISTAL), whose flow chart is given in Figure 5.2.2.1, is applied to solve the related distribution problem ($L2_{ik}$). This algorithm based on the shortest path theory finds the optimal solution in most cases with very little computational effort; even when it can't find the optimal, it provides a tight upper bound. At each period, the temporary decision of "when" and "to which depot" the plant should ship the demand of that period and those of the preceeding periods is made. Let F_t be the minimum cost of satisfying demands of periods 1,2...,t, and f_{lt} be the minimum cost of satisfying the demands of l, l+1,...,t by sending from the manufacturer in period l. Hence there are t possible alternatives of "when to ship" for period t. The alternative with the minimum cost is found by the recursive relationship given by (5.2.2.2).

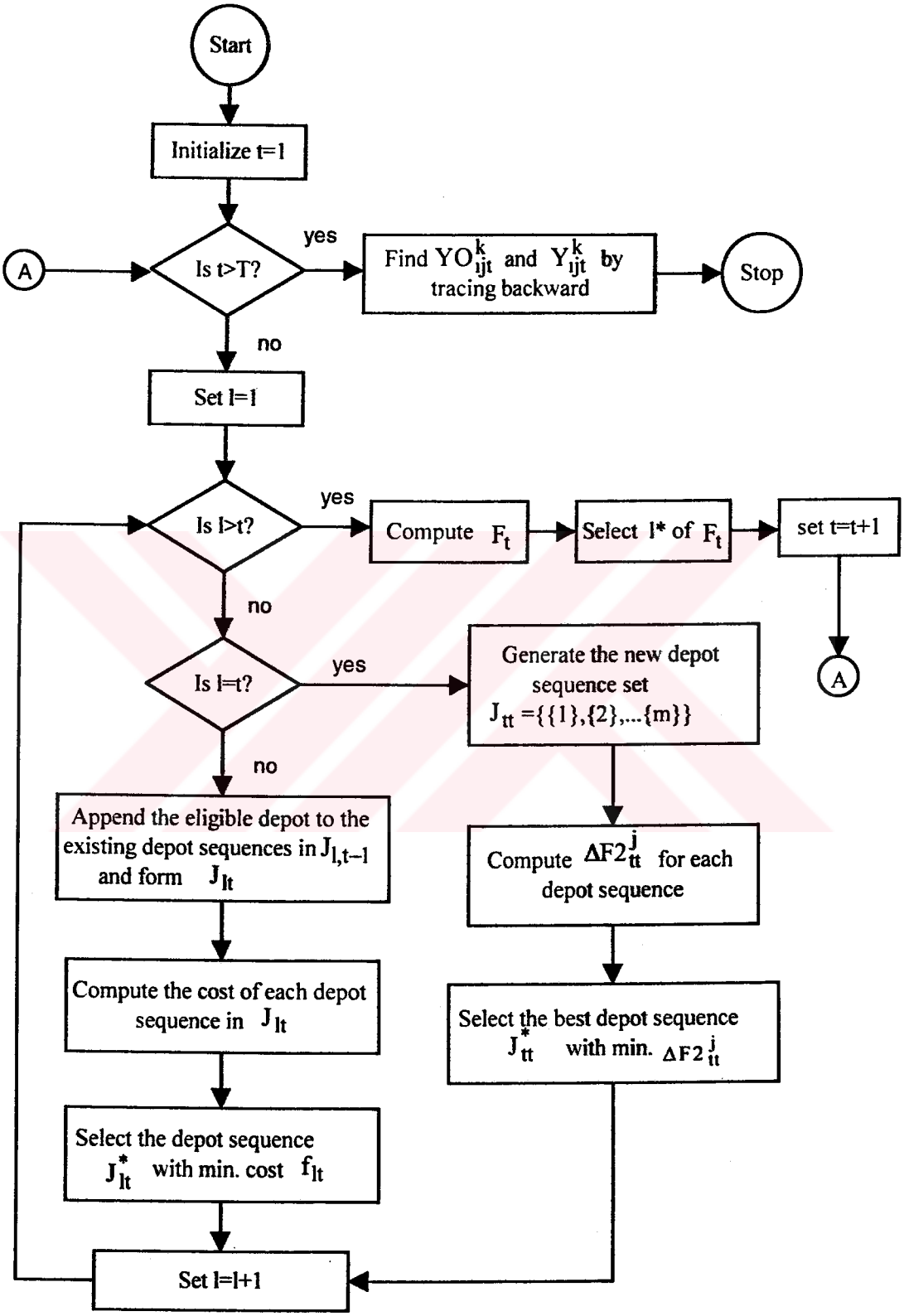

FIGURE 5.2.2.1. Flow chart of DISTAL

$$F_t = \min_{1\le l\le t}\{F_{l-1} + f_{lt}\} \qquad (5.2.2.2)$$

where $F_0 = 0$. At each possible alternative l, we use the previous decision and the information available on the set of alternative depot sequences formed in period t-1 for shipping demands of l, l+1, ..., t-1 in period l ($1\le l\le t-1$) respectively, which is denoted by $J_{l,t-1}$. Here the cost of each depot sequence in $J_{l,t-1}$ is denoted by $f^e_{l,t-1}$. Let ΔFl^j_{lt} be the cost of shipping d^k_{it} in period l from the depot j which is in the existing depot sequence. So

$$\Delta Fl^j_{lt} = (co_{ijl} + \sum_{r=l}^{T} u_{ir} - \sum_{r=l}^{T} ho_{ir})d^k_{it} + \sum_{r=l}^{t-l}(h_{ijr}d^k_{it}) + f^k_{ijt} + c^k_{ijt}d^k_{it} \qquad (5.2.2.3)$$

Let $\Delta F2^j_{lt}$ be the cost of shipping d^k_{it} in period l from the depot j which is not in the existing depot sequence. So

$$\Delta F2^j_{lt} = \Delta Fl^j_{lt} + fo_{ijl} \qquad (5.2.2.4)$$

J^*_{lt} is defined to be the best depot sequence of shipping demands of l,l+1,...,t in l, L_{lt} be the set of depot sequences other than J^*_{lt}, and L^e_{lt} be the e^{th} ordered depot sequence in L_{lt}. The new set of depot sequences J_{lt} is formed by appending the "eligible depot" to each depot sequence of $J_{l,t-1}$. "Eligible depot" is defined to be the depot which satisfies the minimum cost among all possible alternatives of depots which are in $L^e_{l,t-1}$ but not in

$J^*_{l,t-1}$ for each $L^e_{l,t-1}$. If such a depot does not exist, then we delete that sequence from J_{lt}. For $J^*_{l,t-1}$, the "eligible depot" includes all m depots. In this manner, the number of depot sequences to be kept in J_{lt} is reduced.

If l is equal to t, which implies that shipping d^k_{it} is being considered in period t, since distribution decision in t arises in t for the first time, there is no previous information available to reduce the depot domain. Hence J_{lt} is formed such that it covers all of the m depots in the model. For this case (l=t), f_{lt} and J^*_{lt} are determined by:

$$f_{lt} = \min_{1\le j\le m}\left\{\Delta F2^j_{lt}\right\} \tag{5.2.2.5}$$

$$J^*_{lt} = \left\{\arg\min\left\{\Delta F2^j_{lt} \setminus j \in \{1,..,m\}\right\}\right\} \tag{5.2.2.6}$$

If l is not equal to t, the following three cases may occur in the calculation of f_{lt}:

Case1: f_{lt} may belong to a depot sequence generated by a depot sequence of $J_{l,t-1}$ other than $J^*_{l,t-1}$. Then

$$f_{lt} = \min_{1\le e\le|L|}\left\{f^e_{lt}\right\}$$

$$= \min_{1\le e\le|L|}\left\{f^e_{l,t-1} + \min\left\{\Delta Fl^j_{lt} \setminus j \in L^e_{l,t-1}, j \notin J^*_{l,t-1}\right\} \setminus L^e_{l,t-1} \in L_{lt}\right\} \tag{5.2.2.7}$$

Case2: f_{lt} may belong to a depot sequence generated by $J^*_{l,t-1}$ and the newly appended depot is already in $J^*_{l,t-1}$. Then

$$f_{lt} = f_{l,t-1} + \min\left\{\Delta F1^j_{lt} \setminus j \notin J^*_{l,t-1}, j \in \{1,..,m\}\right\} \tag{5.2.2.8}$$

Case3: f_{lt} may belong to depot sequence generated by $J^*_{l,t-1}$ and the newly appended depot is not in $J^*_{l,t-1}$. Then,

$$f_{lt} = f_{l,t-1} + \min\left\{\Delta F1^j_{lt} \setminus j \notin J^*_{l,t-1}, j \in \{1,..,m\}\right\} \tag{5.2.2.9}$$

In short, f_{lt} is determined by:

$$\begin{aligned} f_{lt} = \min\Bigg\{ & \min_{1\le e\le |L|}\left\{f^e_{l,t-1} + \min\left\{\Delta F1^j_{lt} \setminus j \in L^e_{l,t-1}, j \notin J^*_{l,t-1}\right\} \setminus L^e_{l,t-1} \in L_{lt}\right\}, \\ & f_{l,t-1} + \min\left\{\Delta F1^j_{lt} \setminus j \in J^*_{l,t-1}\right\}, \\ & f_{l,t-1} + \min\left\{\Delta F2^j_{lt} \setminus j \notin J^*_{l,t-1}, j \in \{1,..,m\}\right\}\Bigg\} \end{aligned} \tag{5.2.2.10}$$

J^*_{lt} is given by:

$$J^*_{lt} = \begin{cases} L^{e^*}_{l,t-1} + \left\{ \arg\min \left\{ \Delta F1^j_{lt} \setminus j \in L^{e^*}_{l,t-1}, j \notin J^*_{l,t-1} \right\} \right\}, & \text{if case1 determines the min.} \\ J^*_{l,t-1} + \left\{ \arg\min \left\{ \Delta F1^j_{lt} \setminus j \in J^*_{l,t-1} \right\} \right\}, & \text{if case2 determines the min.} \\ J^*_{l,t-1} + \left\{ \arg\min \left\{ \Delta F2^j_{lt} \setminus j \notin J^*_{l,t-1}, j \in \{1,..,m\} \right\} \right\}, & \text{if case3 determines the min.} \end{cases} \quad (5.2.2.11)$$

where $L^{e^*}_{l,t-1}$ is the depot sequence of $L_{l,t-1}$ generating the depot sequence of J_{lt} with the minimum cost.

5.3. Subgradient Optimization Procedure

The next issue is to solve the dual problem of (L) which is denoted by LR with the Lagrangean dual value as

$$Z(LR)=\sup Z(u), \quad (5.3.1)$$

where $u \geq 0$. As it is indicated before, subgradient optimization is implemented to optimize the Lagrangean dual value. The overall subgradient procedure given in Figure 5.3.1 is explained as follows:

Step 1: Set u^0, initial Lagrange multipliers, r, the measure which counts the number of iterations without any improvement of Z(LR), and Z_{lower} to zero, and Z_{upper} to A, a value which can be determined easily by applying a heuristic approach to (PD').

Step 2: If the maximum number of iterations has been reached, stop. Otherwise, go to Step 3.

Step 3: Solve the decomposed two-level distribution problem ($L2_{ik}$) for each (i,k) pair to determine YO^k_{ijt}, Y^k_{ijt} and I^k_{ijt} values.

Step 4: Solve the decomposed production problem ($L1_t$) for each period to determine X_{it} and IO_{it} values.

Step 5: Compute $Z^k(LR)$ as given below:

$$Z^k(LR) = \sum_{i=1}^{I}\sum_{k=1}^{n} Z(L2_{ik}) + \sum_{t=1}^{T} Z(L1_t) + IO_{i0}\left(\sum_{t=1}^{T}\sum_{i=1}^{I} ho_{it} - \sum_{t=1}^{T}\sum_{i=1}^{I} u_{it}\right) \qquad (5.3.2)$$

Step 6: If Z(LR) has improved, go to Step 8. Otherwise, go to Step 7.

Step 7: Increase r by one. If r is equal to R1, let r equal to zero and reduce λ. Otherwise, go to Step 8.

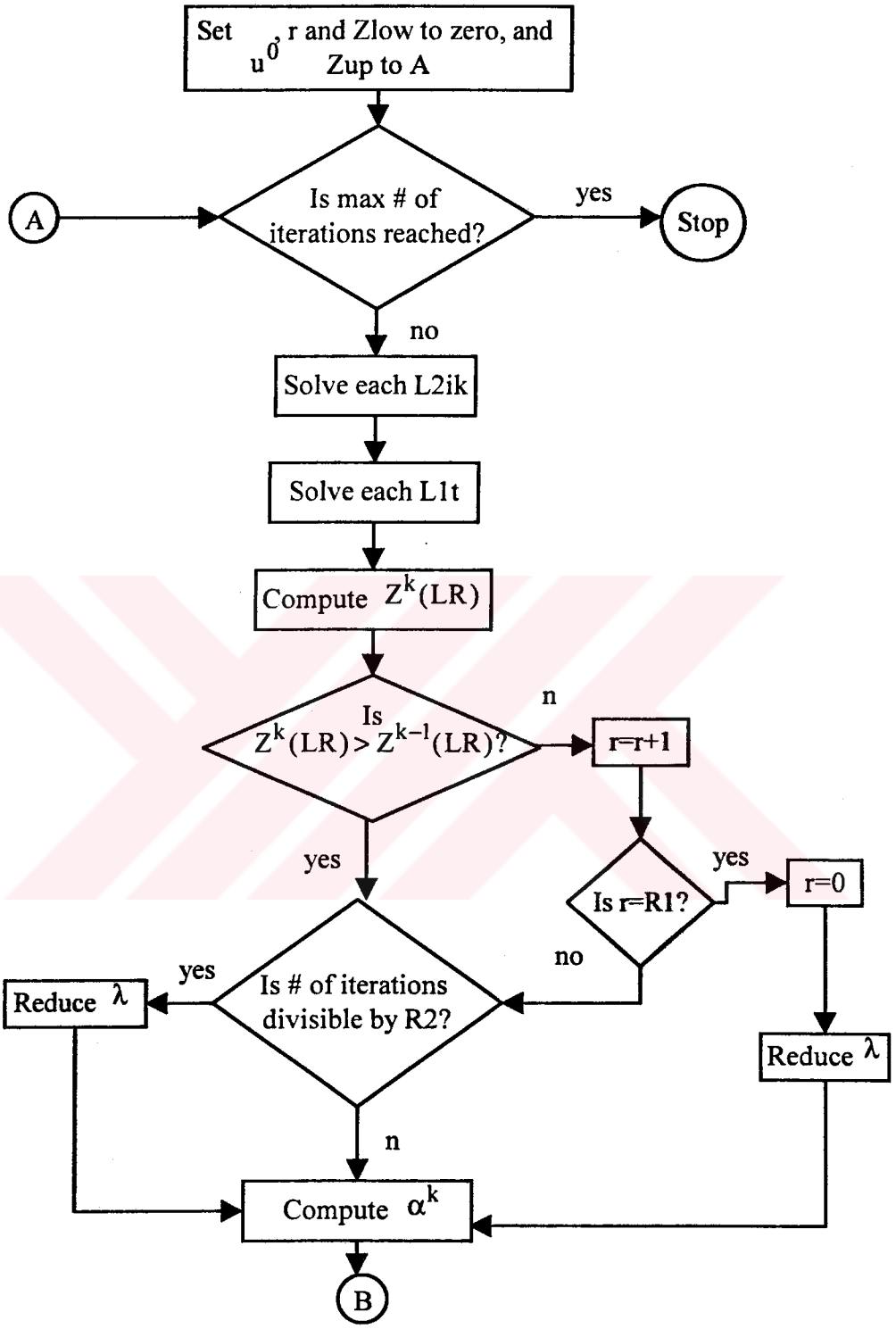

FIGURE 5.3.1. Subgradient optimization procedure to solve problem L

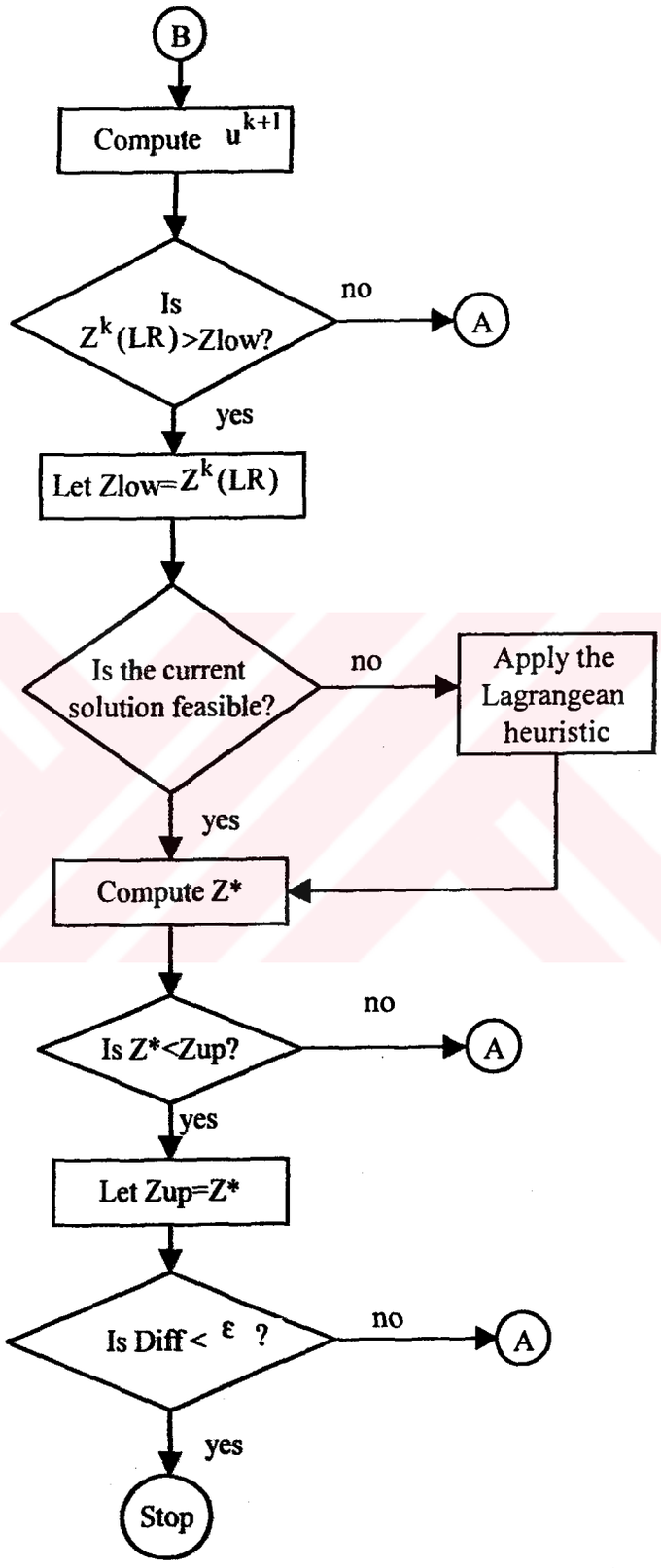

FIGURE 5.3.1. Subgradient optimization procedure to solve problem L (continued)

Step 8: If the number of iterations is divisible by a predetermined number R2, reduce λ. Otherwise, go to Step 9.

Step 9: Compute α^k as shown below :

$$\alpha^k = \lambda^k \frac{\left(Z^k_{upper} - Z^k(LR)\right)}{\|\gamma\|^2} \tag{5.3.3}$$

where $\|\ .\ \|$ denotes the Euclidean norm and

$$\|\gamma\|^2 = \sum_{i=1}^{I}\sum_{t=1}^{T}\left(\sum_{t=1}^{t}\sum_{j=1}^{m}\sum_{k=1}^{n} YO^k_{ijr} - IO_{i0} - \sum_{r=1}^{t} X_{ir}\right)^2 \tag{5.3.4}$$

Step 10: Compute u^{k+1} as shown below:

$$u^{k+1}_{it} = u^k_{it} + \alpha^k \gamma^k_{it} \qquad \forall i,t \tag{5.3.5}$$

where γ^k_{it} is the subgradient of (LR) belonging to the (i,t) pair and it is computed as follows:

$$\gamma_{it} = \sum_{r=1}^{t}\sum_{j=1}^{m}\sum_{k=1}^{n} YO_{ijt}^{k} - \sum_{r=1}^{t} X_{ir} - IO_{i0} \qquad \forall i,t \qquad (5.3.6)$$

Step 11: If $Z^k(LR) > Z_{lower}$, set $Z_{lower} = Z^k(LR)$ and go to Step 12. Otherwise, go to Step 2.

Step 12: Check if the current solution is feasible to the original problem. If it is not feasible, apply the Lagrangean heuristic to make it feasible. Go to Step 13.

Step 13: Compute the feasible solution Z^*.

Step 14: If $Z^* < Z_{upper}$, let $Z_{upper} = Z^*$ and go to Step 15. Otherwise, go to Step2.

Step 15: If Diff $< \varepsilon$, then stop. Otherwise, go to Step 2. Here $\varepsilon > 0$ is a sufficient percentage of closure of Z_{upper} to Z_{lower} and Diff is the duality gap and is computed as in (5.3.7).

$$\text{Diff} = \frac{(Z_{upper} - Z_{lower})}{Z_{upper}} \qquad (5.3.7)$$

There are some important issues to be mentioned in the subgradient optimization procedure:

(a) In this study, the initial value of Z_{upper} denoted by A is calculated by executing the subgradient optimization once. In other words, it is equal to Z^* found in the first iteration of the subgradient optimization procedure where $u_{it}=0$ $\forall i,t$. This is similar to the situation where production and distribution problems are decomposed hierarchically into

two subproblems such that the distribution problem is solved first and then the production subproblem is solved based on the solution of the distribution problem.

(b) R1 and R2 mentioned in the procedure are the two control parameters used to reduce the step size whenever it is necessary and are taken to be both ten. Initially λ^0 is two, and reducing λ is done by halving it. So in this study, λ^k is halved if the objective function does not improve in ten iterations and every ten iterations regularly.

(c) The algorithm is terminated either when a given number of iterations is reached or whenever Diff $< \varepsilon$ for some $\varepsilon>0$ whichever occurs first. In this study, the procedure is run for at most 100 iterations and ε is taken to be 0.01.

5.4. The Lagrangean Heuristic

In this procedure there is no guarantee that the solution of the dual problem (LR) generates a solution feasible to the original problem (PD'). In fact, it is a rare situation that the dual solution is feasible to (PD'). This is the result of the duality gap phenomenon which is just capable of providing a lower bound for the optimal value of (PD'). However the dual solutions are near-feasible, and one can easily generate a feasible solution out of it. Thus the dual solution is input into a Lagrangean heuristic to provide a feasible solution with an objective function value which can be used as the upper bound for the original problem.

The infeasibility in this problem may arise from the inconsistency between the solution of (L1) and the solution of (L2). Since dualizing constraints (4.2.5) decouples the production and distribution decisions and the respective production and distribution decisions are made independently, the production quantities may not be sufficient to meet

distribution requirement of product i in period t determined by $\sum_{j=1}^{m}\sum_{k=1}^{n} YO_{ijt}^{k}$. Then it will be necessary to shift production and/or distribution decisions so as to restore consistency. Here it is intended to preserve the distribution decisions as long as production capacity permits, so a hierarchical interdependency is imposed by placing the distribution subproblem in the top level and transmitting the instructions to the production subproblem. In fact the heuristic is designed to enhance a "pull" nature in the system.

The Lagrangean heuristic developed in this study specifically consists of two procedures. In the first procedure (DISTFEAS), it is checked out whether there is sufficient production capacity to meet the distribution requirements,

$$\sum_{r=1}^{t} A_r \geq \sum_{r=1}^{t}\sum_{i=1}^{I}\left[\sum_{j=1}^{m}\sum_{k=1}^{n} YO_{ijr}^{k}\right] a_i \qquad \forall t \qquad (5.4.1)$$

If (5.4.1) is not satisfied at some period t, then some distribution requirement should be shifted to a later period. To achieve this, it is required to decide which YO_{ijl}^{k}, $l \in [1, t]$, should be decreased and how the shifted quantity should be redistributed over [t+1, T]. The algorithm first determines a particular (i,k) pair from which shifting future demand requirements is possible, by ranking all possibilities and choosing the combination which results in the maximum saving given by W_{ik}. Then the distribution problem is resolved for the reduced problem of (i,k) pair over [t+1, T]. This procedure is repeated recursively by eliminating infeasibilities step-by-step for smaller problems. The procedure (DISTFEAS) whose flow chart is presented in Figure 5.4.1 is given below:

For t=1 to T, **do**

begin

if $\sum_{r=1}^{t}\sum_{i=1}^{I}\sum_{k=1}^{K}\sum_{j=1}^{J} YO_{ijr}^{k} a_i > \sum_{r=1}^{t} A_r$ for some t **then**

begin

set $W_{ik} = 0$ and $q_{ik} = 0$ for all i,k

$D_t^{ik} = \{t' \setminus l_{it'}^{k} \le t, t' > t\}$, for all i, k

$E_t = \{(i,k) \setminus D_t^{ik} \neq \emptyset, i \in I, k \in K\}$

For each (i,k) in E_t, **do**

begin

$t^0 = l_{ir}^{k}$, for all $r \in D_t^{ik}$

M_j=1, for all j

For each t′ such that $t^0 \le t' \le t$, **do**

begin

$j = b_{it'}^{k}$

set M_j=0

end

For each r in D_t^{ik}, **do**

begin

$j = b_{itr}^{k}$

$W_{ik} = W_{ik} + fo_{ijt^0} M_j + d_{ir}^{k}(co_{ijt^0} - \sum_{n=t^0}^{T} ho_i) + \sum_{n=t^0}^{t} d_{ir}^{k} h_{ijn} + d_{ir}^{k} c_{ijr}^{k} + f_{ijr}^{k}$

set M_j=0

$q_{ik} = q_{ik} + d_{ir}^{k}$

end

$W_{ik} = W_{ik} / (q_{ik} a_i)$

end

While $\sum_{r=1}^{t}\sum_{i=1}^{I}\sum_{k=1}^{K}\sum_{j=1}^{m} YO_{ijr}^{k} a_i > \sum_{r=1}^{t} A_r$ **do**

begin

$(i',k') = \arg\max\{W_{ik} \setminus (i,k) \in E_t\}$

$t^0 = l^{k'}_{i'r}$, for all $r \in D^{i'k'}_t$

For each r in $D^{i'k'}_t$, **do**

begin

$j = b^{k'}_{i'r}$

$YO^{k'}_{i'jt^0} = YO^{k'}_{i'jt^0} - d^{k'}_{i'r}$

For each t′ such that $t^0 \le t' \le t$, **do**

begin

$I^{k'}_{i'jt'} = I^{k'}_{i'jt'} - d^{k'}_{i'r}$

end

end

set $YO^{k'}_{i'jt'} = Y^{k'}_{i'jt'} = I^{k'}_{i'jt'} = 0$, for all $t' \ge t+1$

run DISTAL for (i',k') starting from t+1

$E_t = E_t - \{(i',k')\}$

end

end

end

Here the parameters are defined as follows:

W_{ik} = a pessimistic measure of savings which can be obtained by shifting the planned shipment of future demand requirements of (i,k)th product-customer pair to a later period.

q_{ik} = total quantity of (i,k)th product-customer pair which can be shifted to later periods.

D^{ik}_t = the set of future periods of (i,k)th product-customer pair whose demand requirements are to be shipped from the manufacturer before or equal to the current period t.

l^k_{it} = the shipment time of demand d^k_{it} from the manufacturer.

E_t = the set of product-customer pairs whose future demand is shipped before or equal to the current period t.

t^0 = the period before or equal to the current period t where there is an early shipment of demand because of the lotsizing effect.

M_j = a binary variable indicating if total cost can be reduced by shifting some future demand from depot j.

b_{it}^k = the depot to which the manufacturer ships the demand d_{it}^k.

(i', k') =product-customer pair which has the maximum W_{ik} at a particular period t, $(i,k) \in E_t$.

Once it is guaranteed that the solution of the distribution problem satisfies the production capacity constraints, the second part of the algorithm aims to match the production decisions to distribution solution. If an inconsistency arises in period t for the first time, first it is attempted to resolve the inconsistency in the same period; however if capacities do not permit, some free capacity is sought by searching previous periods in the order (t-1, t-2, ..,1). Here the main idea is to allocate the closest free capacity to a product with the minimum cost increase measured by V_i. The second procedure (PROFEAS) whose flow chart is given in Figure 5.4.2 is summarized as follows:

For t=1 to T, **do**

begin

$$IO_{it} = IO_{i0} + \sum_{r=1}^{t} X_{ir} - \sum_{r=1}^{t} \sum_{j=1}^{m} \sum_{k=1}^{n} YO_{ijr}^k \text{, for all i}$$

$$S_t = \{ i \setminus IO_{it} < 0 \}$$

While S_t is not empty **do**

begin

For each i in S_t, **do**

begin

$deficit_i = -IO_{it}$

l=t

While ($deficit_i$>0) **do**

begin

if $\text{excess}_l \geq a_i$ **then**

begin

$q=\min\{\text{excess}_l/a_i, \text{deficit}_i\}$

$V_i=V_i+q(p_{il}+\sum_{r=l}^{T} ho_{ir})+(1-Z_{il})s_{il}$

$\text{deficit}_i=\text{deficit}_i-q$

end

if $\text{load}_l \geq a_i$ and $\text{deficit}_i>0$ **then**

begin

$q=\min\{\text{load}_l/a_i, \text{deficit}_i\}$

$d=\text{item}_l$

$q_d=\lceil q(a_i/a_d)\rceil$

$V_i=V_i+q(p_{il}+\sum_{r=l}^{T} ho_{ir})+(1-Z_{il})s_{il}$

$V_i=V_i-q_d(p_{dl}+\sum_{r=l}^{T} ho_{dr})$

if $q_d=X_{dl}$ **then**

$V_i=V_i-s_{dl}$

$\text{deficit}_i=\text{deficit}_i-q$

end

$l=l-1$

end

$V_i=V_i/((-IO_{it})a_i)$

end

$i'=\arg\min\{V_i \setminus i \in S_t\}$

Update all required X_{it}, IO_{it}, excess_t, load_t, item_t

$S_t=S_t-\{i'\}$

end

end

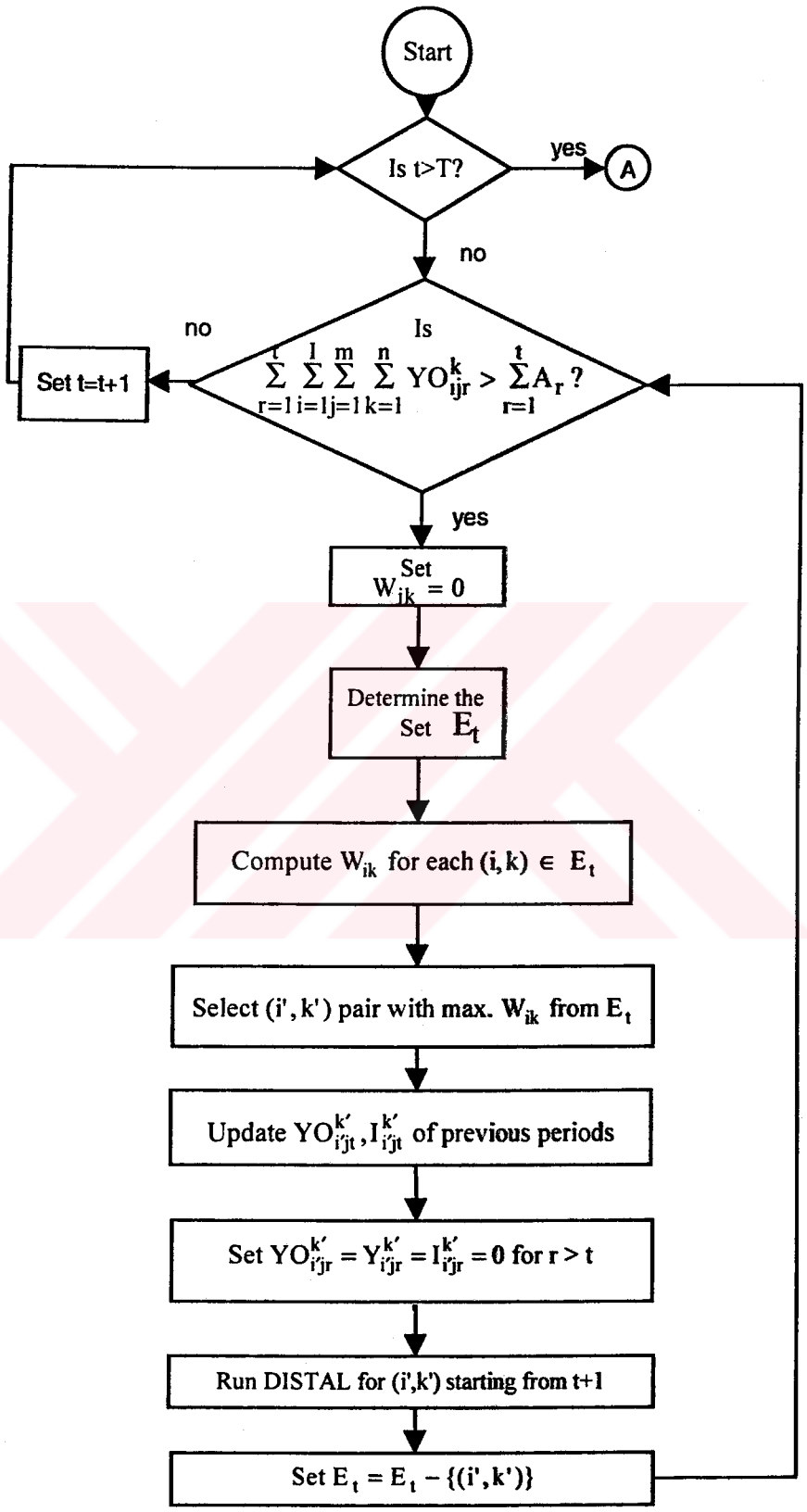

FIGURE 5.4.1. Flow chart of DISTFEAS

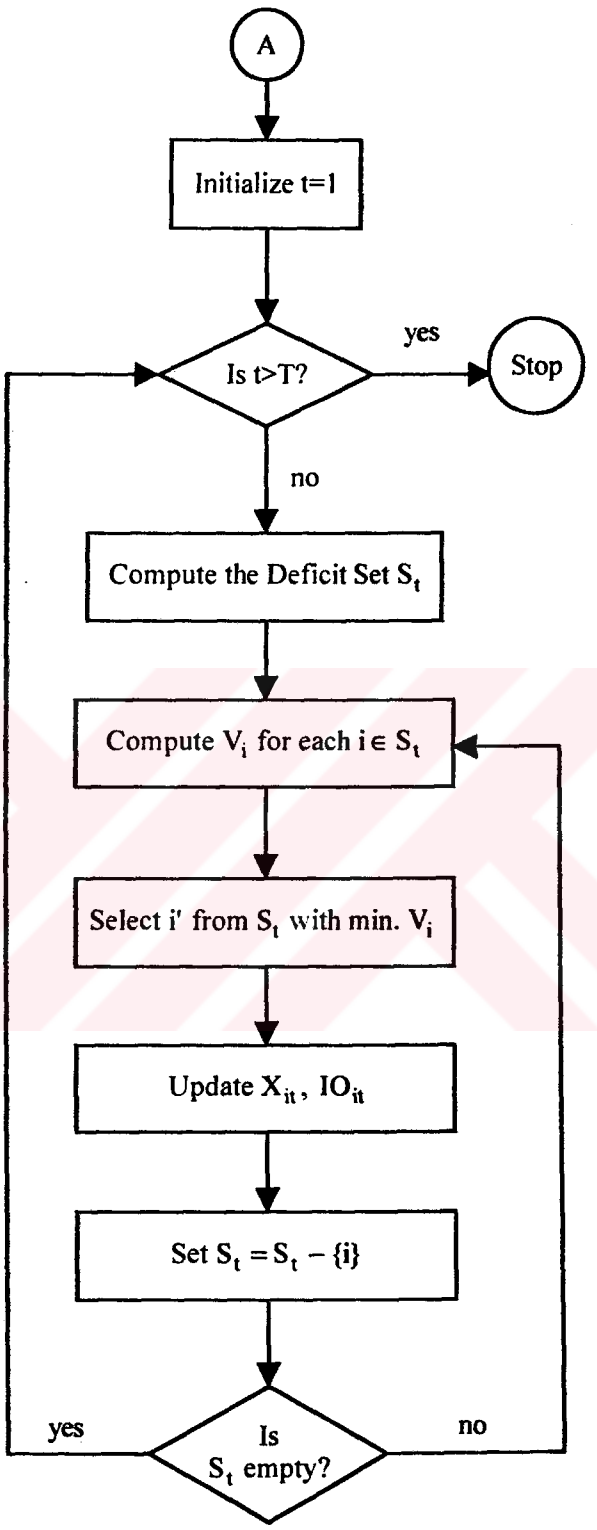

FIGURE 5.4.2. Flow chart of PROFEAS

Here the parameters are defined as follows:

S_t = the set of products with shortages in t.

$deficit_i$ = the shortage of product i.

$excess_t$ = unused capacity in t.

q = quantity of any product having shortage which is no more shortage.

v_i = added cost per capacity usage of product i, $i \in S_t$.

$load_t$ = capacity used for over-production in t.

$item_t$ = product over-produced in t.

q_d = reduced amount of an over-produced product for another product having shortage.

i' = product which has the minimum v_i, $i \in S_t$

5.5. Interpretation of Dual Variables

The Lagrange multipliers play an important role in the information exchange between the production and distribution decisions. After applying Lagrangean relaxation to the constraint (4.2.5) in the aggregate model (PD') which couples production and distribution, the problem naturally decomposes into production and distribution decision subproblems.

The sum of Lagrange multipliers from t to T, $\left\{ \sum_{r=t}^{T} u_{ir} \right\}$ split the production cost $p_{it} + \sum_{r=t}^{T} ho_{ir}$ between the production submodel L1 and the distribution submodel L2. So, the determination of the appropriate multipliers, which causes the appropriate allocation of the production cost between the production submodel and distribution submodel may be interpreted as a feedback process between the two subproblems. During the determination

stage of the Lagrange multipliers $\{u_{it}\}$, the subgradient optimization is implemented to coordinate the information flow between the two subproblems.

The dual problem of L is solved iteratively, where at each iteration, the Lagrange multipliers $\{u_{it}\}$are updated. The revision of the Lagrangean multipliers depends upon the current degree of inconsistency between the two subproblems. For a given value of u_{it} in iteration k, both the production subproblem L1 and distribution subproblem L2 are solved; based on these solutions the set of Lagrange multipliers is revised and the next iteration k+1 is performed with the modified values of u_{it}.

In this manner, the solutions of the production and distribution subproblems are continually revised based on the information from the other subproblem which is carried by Lagrange multipliers $\{u_{it}\}$, until a solution is obtained which matches the production decisions with the distribution decisions.

So, the Lagrange multipliers coordinate the information flow between the two subproblems at each iteration and it is tried to determine the appropriate values for the Lagrange multipliers by trying to maximize the dual problem and also reach the feasible solution where no inconsisteny exists between the subproblems.

6. A TABU SEARCH ALGORITHM FOR THE VEHICLE ROUTING PROBLEM

After solving the aggregate production-distribution problem of upper level in the hierarchical decomposition, vehicle routing problem will be addressed based on the upper level decisions. A new tabu search heuristic is developed to solve the vehicle routing problem with both capacity and tour length constraints. The heuristic generates a set of neighbours by exchanging a set of vertices between their respective current routes and moving to the feasible non-tabu neighbour with the best objective function value. In fact during the neighbourhood construction procedure, a combination of traditional improvement techniques are used simultaneously.

6.1. Problem Formulation

The Vehicle Routing Problem (VRP) can be simply stated as the problem of determining optimal routes through a set of locations and defined on a directed graph G=(V,A) where V=(v_o, v_1,.....,v_n) is a vertex set and A=((v,v): v_i, $v_j \in V$, $i \neq j$) is an arc set.Vertex v_o represents a depot where a fleet of N_v vehicles of the same capacity are located. The value of N_v can be either prespecified or free, i.e. bounded above by a constant $N \leq n-1$. All remaining vertices represent customers. A nonnegative (distance/time/cost) matrix $C=(c_{ij})$ is defined on A. Here since $c_{ij}=c_{ji}$ for all (v_i,v_j), the problem is said to be symmetric and arcs are represented by undirected edges. A nonnegative weight d_i is associated with each vertex to represent the customer demand at v_i, and naturally the weight assigned to any route may not exceed the vehicle capacity Q_v. In some problems a further limitation is imposed on the total route duration in addition to the vehicle capacity constraint. In such a case t_{ij} is defined to represent the travel time for each (v_i,v_j) and t_i

represents the service time at any vertex v_i and it is required that the total duration of any route not exceed a preset bound T_v. Thus VRP aims at determining N_v vehicle routes of minimal total cost, each starting and ending at the depot, so that every customer is visited exactly once subject to the above mentioned constraints.

A typical mathematical formulation for the single depot VRP is provided below:

$$\text{Minimize} \quad \sum_i \sum_j \sum_v c_{ij} X_{ij}^v \tag{6.1.1}$$

subject to

$$\sum_i \sum_v X_{ij}^v = 1 \qquad \forall j \tag{6.1.2}$$

$$\sum_j \sum_v X_{ij}^v = 1 \qquad \forall i \tag{6.1.3}$$

$$\sum_i X_{ip}^v - \sum_j X_{pj}^v = 0 \qquad \forall p, v \tag{6.1.4}$$

$$\sum_i d_i \left(\sum_j X_{ij}^v \right) \leq Q_v \qquad \forall v \tag{6.1.5}$$

$$\sum_i t_i^v \sum_j X_{ij}^v + \sum_i \sum_j t_{ij}^v X_{ij}^v \leq T_v \qquad \forall v \qquad (6.1.6)$$

$$\sum_{j=1}^{n} X_{0j}^v \leq 1 \qquad \forall v \qquad (6.1.7)$$

$$\sum_{i=1}^{n} X_{i0}^v \leq 1 \qquad \forall v \qquad (6.1.8)$$

$$X_{ij}^v \in S \qquad \forall i, j, v \qquad (6.1.9)$$

In this formulation the decision variables X_{ij}^v are binary variables indicating if arc(i,j) is traversed by vehicle v. The objective function of distance/cost/time minimization is expressed by (6.1.11). In this study the locations of the vertices are specified by their respective coordinates and Euclidean distances are used without loss of generality. Constraints (6.1.2) and (6.1.3) together state that each demand vertex be served by exactly one vehicle. It is guaranteed in (6.1.4) that a vehicle leaves the demand vertex it has already entered. Vehicle capacity constraints are expressed by (6.1.5) whereas the limitation on the maximum route duration is given by (6.1.6). Constraints (6.1.7) and (6.1.8) express that vehicle availability not be exceeded. The subtour elimination constraints are defined in (6.1.9) where S can be defined in anyone of the ways found very frequently in literature.

Over the last decade, there have been important advances in the development of exact and approximate algorithms for the VRP which is an NP- hard combinatorial

optimization problem in nature. As far as the exact approaches are concerned, the most significant progress has been made in the design of branch-and-cut and of column generation algorithms. Moreover major advances have taken place in the area of metaheuristics, including simulated annealing, tabu search, genetic search and neural networks. In this study a tabu search algorithm is developed to solve the single-depot VRP arising after the solution of the integrated production-distribution problem discussed in Section 4. Since customer-product pairs (i,k) are already assigned to the depots, a decentralized logistics management suffices to solve a single-depot VRP for each depot in each period. The details of the tabu search are provided in the next section.

6.2. Tabu Search Algorithm

Tabu search is a metastrategy iterative procedure for building extended neighborhood with particular emphasis on avoiding being caught in a local optimum. This global optimization metaheuristic was initially proposed by Glover [50]. It consists of exploring the search space by moving from a solution to its best neighbour even if this results in a deterioration of the objective function value. This way the likelihood of moving out of local optima is increased. The successive neighbours of a solution are generated and their objective function values are examined. To avoid cycling, solutions that were recently examined are forbidden or declared *tabu* for a certain number of iterations. A move made in iteration t is called *tabu* until iteration $(t+\theta)$ where θ is the tabu duration randomly chosen on a prespecified interval. Thus the tabu list is an ordered queue containing forbidden moves; whenever a move is made, it is inserted to the end of the tabu list and the first element from the list is removed. The best admissible move is then chosen as the highest evaluation move in the neighbourhood of the current solution in terms of the objective function value and the tabu restrictions. In some studies the whole neighbourhood is explored and the best non-tabu move is selected; however in some other studies the first feasible non-tabu improving move is selected. Since tabu search permits the disimproving

moves as well, it is necessary to restrain the search only to non-tabu moves. Even an improving move is not accepted if it is forbidden in the tabu list unless it satisfies the *aspiration criterion.*The aspiration criterion is a measure solely designed to override the tabu status of a move if this move leads to a solution better than the best found by the search so far. *Intensification* strategies can be applied to accentuate the search in a promising region of the solution space, so that the moves to the local optimum are intensified. *Diversification* on the other hand can be used to broaden the search into less explored regions by forcing the moves out of the local optimum.

All these concepts are extensively used in the tabu search **DETABA** developed in this study to solve the above mentioned VRP. Generally in the VRP context, it is a common practice to define a neighbour by either the exchange of vertices between different routes,the replacement of vertices into a different route or within its own route, the creation of a new route or the deletion of an existing route. The neighbourhood construction heuristic **TANE** which attempts to improve upon a given solution S is naturally the main heuristic in the tabu search DETABA designed in this research and defines the neighbourhoods by means of interchange mechanisms which includes a combination of 2-Opt moves, vertex reassignments to different routes and vertex interchanges between two routes. Here a given number of neighbours around the current solution are explored and the best non-tabu move is implemented. **TANE** is characterized by a vector of parameters

$$(N_{max},\ \alpha_{max},\ \beta_{max},P,\ \theta_{min},\ \theta_{max}) \qquad (6.2.1)$$

where

N_{max} = maximum number of iterations without any improvement

α_{max} = maximum number of neighbours to be generated in the neighbourhood

β_{max} = maximum number of 2-Opt exchanges

(M,P) = maximum number of vertices to interchanged between the routes in S

$(\theta_{min}, \theta_{max})$ = bounds on the tabu duration

The algorithm **TANE** whose flow chart is given in figure 6.2.1 is as follows:

Step 1: Execute the following steps to generate a neighbour.

Step 1.1: Randomly choose a route and call it R1

Step 1.2: Randomly choose another route and call it R2

Step 1.3: Randomly choose $\mu \leq \min (M, |R1|)$ vertices out of R1.

Step 1.4: Randomly choose $\pi \leq \min (P, |R2|)$ vertices out of R2.

Step 1.5: Try to insert the chosen π vertices into R1 and μ vertices into R2 by using the insertion procedure.

Step 1.6: If constraints are violated after insertion, go to step 1.3 to generate new μ and π vertices. Otherwise, continue.

Step 1.7: Apply the 2-Opt procedure to R1 and R2 independently for β_{max} times.

Step 1.8: Compute the objective function value of the final situation.

Step 2: Repeat the above steps for α_{max} times.

Step 3: Implement the non-tabu move with the best objective function value.

In **DETABA**, the tabu restriction is defined so as to avoid the reassignment of previously moved vertices to the original routes they have already moved from. If at iteration k vertex $v_1 \in$ R1 and $v_2 \in$ R2 are exchanged (i.e. v_1: R1$\rightarrow$R2 and v_2: R2$\rightarrow$R1), it is forbidden to put both v_1 in R1 and v_2 in R2 again during iterations (k+1,.....,k+θ) where θ is the tabu duration chosen randomly from (θ_{min}, θ_{max}). If all the moves in the whole neighbourhood becomes tabu and none satisfies the aspiration criterion, the algorithm chooses the oldest tabu move and proceeds accordingly. In the solution improvement phase

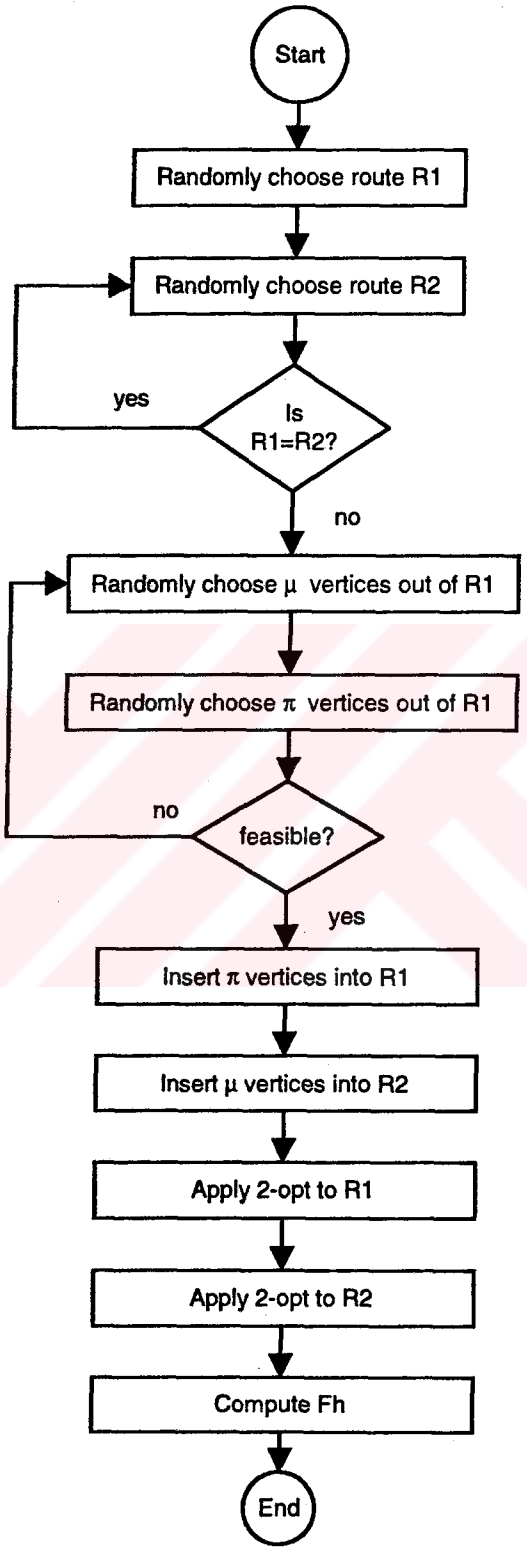

FIGURE 6.2.1. Flow chart of TANE

periodically, in fact every 5n moves the algorithm **DETABA** checks if a new best solution has been found. If not, it goes back to the last best solution.

The other key issues to be specified in **DETABA** are the methods used to construct the initial solutions, the insertion procedure, the 2-Opt improvement procedure and the intensification strategy,each of which will be discussed in detail.

6.2.1. Construction of an initial solution

It is necessary to generate $\sqrt{n}/2$ solutions to initialize the main search process in **DETABA**. Thus two methods are proposed to construct these in a rather random manner without much emphasis on the optimality. These solutions will then be improved by the main algorithm and the best among them will be chosen as the initial solution to the main body of the search itself.

Method 1:

Step 1. Generate a random vertex (v_i) and form the vertex sequence

$$(v_o, v_i, v_{i+1}, \ldots\ldots\ldots, v_n, v_1, v_2, \ldots\ldots\ldots, v_{i-1}) \qquad (6.2.1.1)$$

Step 2. Starting with v_o, create routes by following the above vertex sequence such that the first vehicle contains all cities starting from the first city in the sequence $\leq Q_v$, and repeat this process until all vertices have been included into the routes.

Method 2:

Step 1: Generate a subroute consisting of only vertex v_i chosen randomly.

Step 2: Find v_k such that c_{ik} is minimal and form the subroute (v_o, v_i, v_k, v_o)

Step 3: Find some v_k not already in the subroute closest to any node in the subroute which will not violate Q_v. If there is no such a vertex, generate another route by another random vertex chosen from among the remaining ones.

Step 4: Find the arc(i,j) in the subroute which minimizes $(c_{ik} + c_{kj} - c_{ij})$, and insert v_k between v_j and v_i.

6.2.2. The Insertion Procedure

In Steps 1.5 and 1.6 of the neighbour search procedure **TANE**, it becomes necessary to insert several vertices into a given route simultaneously; thus a simple insertion rule is required to place $\{v_i: v_i \in VP\}$ into the route R where VP is a proper subset of V.

Step 1: Find the vertex $v_k \in VP$ with minimum c_{ik} value to any vertex already in the route R.

Step 2: Find the arc (i,j) in the route R which minimizes $(c_{ik} + c_{kj} - c_{ij})$ and insert vertex k between i and j.

Step 3: Repeat Steps 1 and 2 until all the vertices in VR are placed in the route

6.2.3. The 2-Opt Improvement Procedure

A 2-Opt procedure is employed in Step 1.7 of **TANE** in order to intensify the solution improvement in each route locally. Suppose a subtour consists of the following set of S vertices in the given order $S=\{v_o,v_1,v_2,\ldots\ldots,v_k,v_o\}$, and let $X=\{(v_i,v_{i+1});(v_j,v_{j+1})\}$

be a set of two edges in S which are to be replaced with edges $Y=\{(v_i,v_j);(v_{i+1},v_{j+1})\}$ if this replacement will lead to an improvement. Here it is required that all the vertices under consideration be different from each other. Once the set X has been chosen, the set Y is directly determined. In a subtour consisting of k customer vertices and a depot, there are [(k+1)(k-2)/2] possible edge combinations given by the set E. The 2-Opt algorithm can be summarized as follows:

Step 1: For each $X \in E$, calculate the improvement δ_x obtained by replacing X by the associated Y and given by

$$\delta_x=(c_{i,I+1}+c_{j,j+1})-(c_{ij}+c_{i+1,j+1}) \qquad (6.2.3.1)$$

Step 2: Calculate δ_{max} by

$$\delta_{max}=\max\{\delta_x\} \qquad (6.2.3.2)$$

Step 3: If $\delta_{max} > 0$, replace the two edges asociated with δ_{max} and repeat this for all Xin E

6.2.4. Intensification Strategy

Since the search methods possess the risk of being trapped in a local minimum and the possibility of overlooking a global optimum hidden in a deep valley of the objective function, the intensification efforts in **DETABA** are mainly directed as an attempt to characterize the unexplored regions. To do so, during the execution of the solution improvement stage, the solutions which are suspected to indicate such a search direction

are identified as good candidates to initialize the intensification stage and included in the set INS. At the end of the solution improvement step, $\sqrt{n.m}$ solutions with the minimum objective function value among those are selected for intensification. Suppose in iteration k a solution S_k with an objective function value $C(S_k)$ is obtained and in iteration k+1 a neighbour of this solution S_{k+1} is found to be the best admissible move in the neighbourhood generated around the solution S_k, so that the search will move to S_{k+1}. The change in the objective function values, $\Delta_k=C(S_{k+1}) - C(S_k)$, is calculated. If $\Delta_k \leq 0$, then it is obvious that the search is improving towards a local minimum; however if $\Delta_k > 0$, then the search is hill-climbing. Keeping track of Δ_k will provide valuable information as to which solutions worth the intensification efforts. Thus if $\Delta_{k-1} \leq 0$ and $\Delta_k > 0$, then S_k is identified as the indicator of a promising region which may worth further exploration.

At this point it is possible to summarize the modules of the main tabu search **DETABA**:

Step 0 (Initialization): Generate$\sqrt{n}/2$ initial solutions by using one of the methods discussed in the initial solution generation procedure.

Step 1 (First Improvement): Apply the search algorithm to each of the initial solutions with N_{max}=n, α_{max}=5, β_{max}=all, M=2, P=2, θ_{min}=5, θ_{max}=10.

Step 2 (Solution Improvement): Starting with the best solution obtained in Step 1, apply the search algorithm with N_{max}=50n, α_{max}=n.m/2, β_{max}=all, M=2, P=2, θ_{min}=10, θ_{max}=20.

Step 3 (Intensification): Apply the search algorithm to $\sqrt{n.m}$ solutions with the minimum objective function value from among the solutions determined in Step 2 and included in the set INS with N_{max}=n, α_{max}=n.m/2, β_{max}=all, M=2, P=2, θ_{min}=5, θ_{max}=10..

A lower bound on the number of vehicles m =$\sum$di / Q is used as a measure of the minimum number of routes that can be obtained in any iteration and used in the

determination of α_{max}. It is also used as a normalizing factor in the computation of the number of solutions to be improved in the intensification stage.

To simplify DETABA, subprocedures for neighbourhood search and solution improvement are formed. Flow charts of solution improvement, neighbourhood search and DETABA are provided in figures 6.2.2-6.2.4.

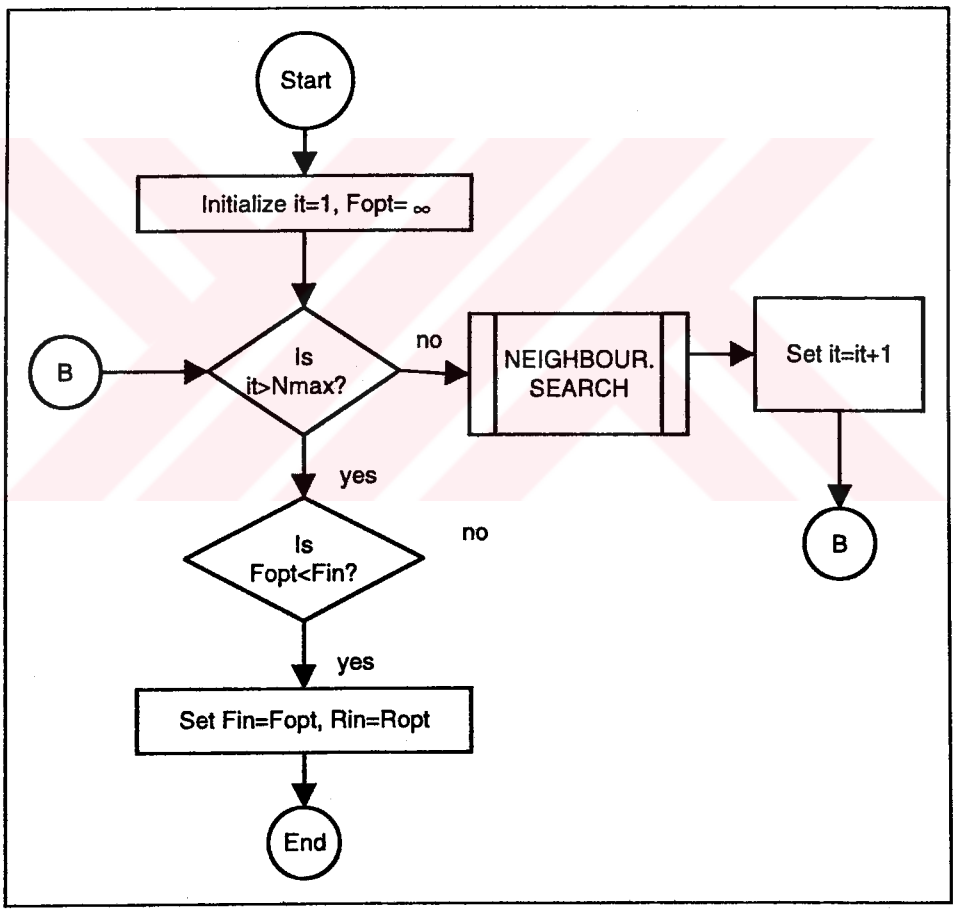

FIGURE 6.2.2. Flow chart of solution improvement

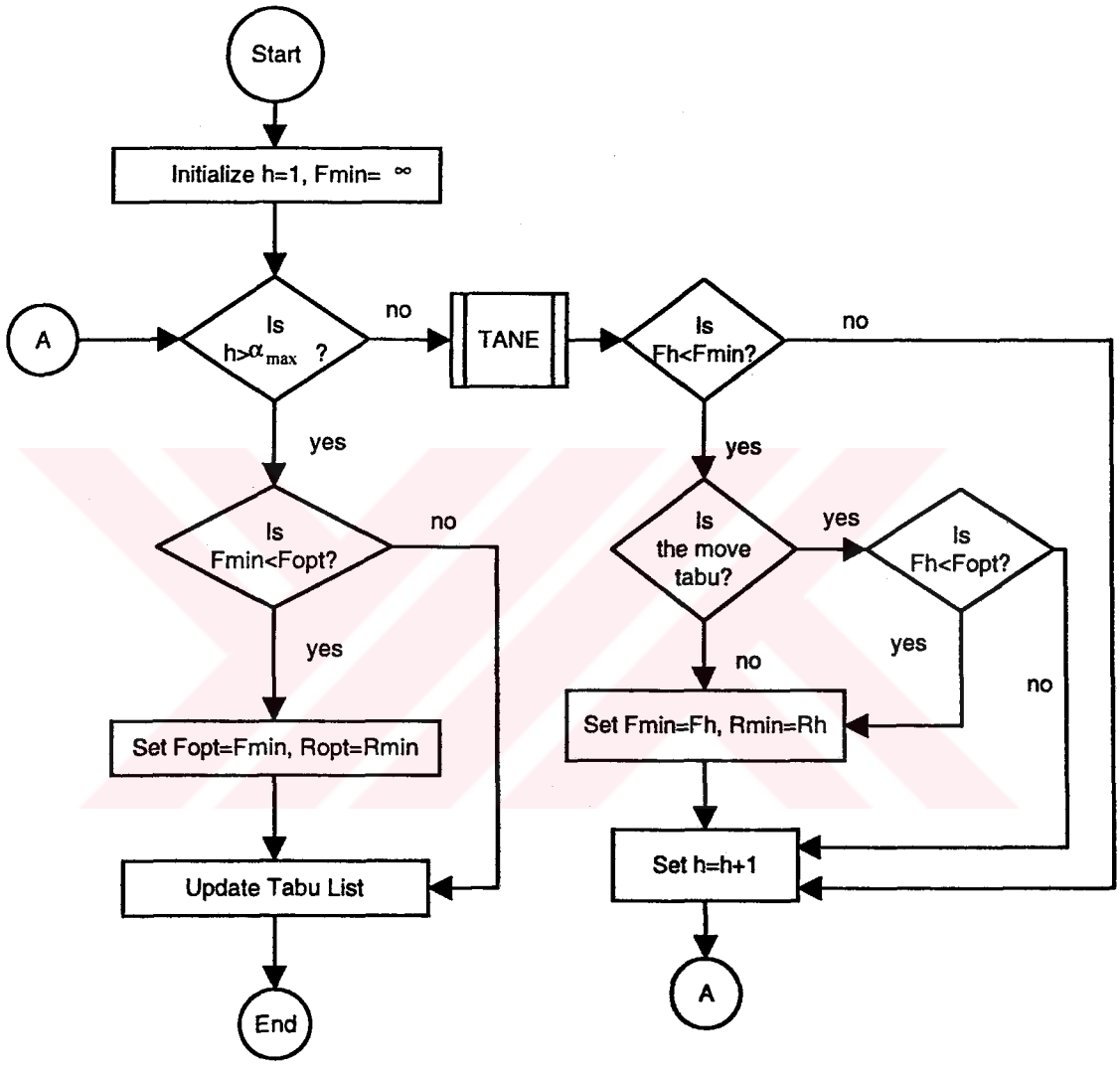

FIGURE 6.2.3. Flow chart of neighbourhood search

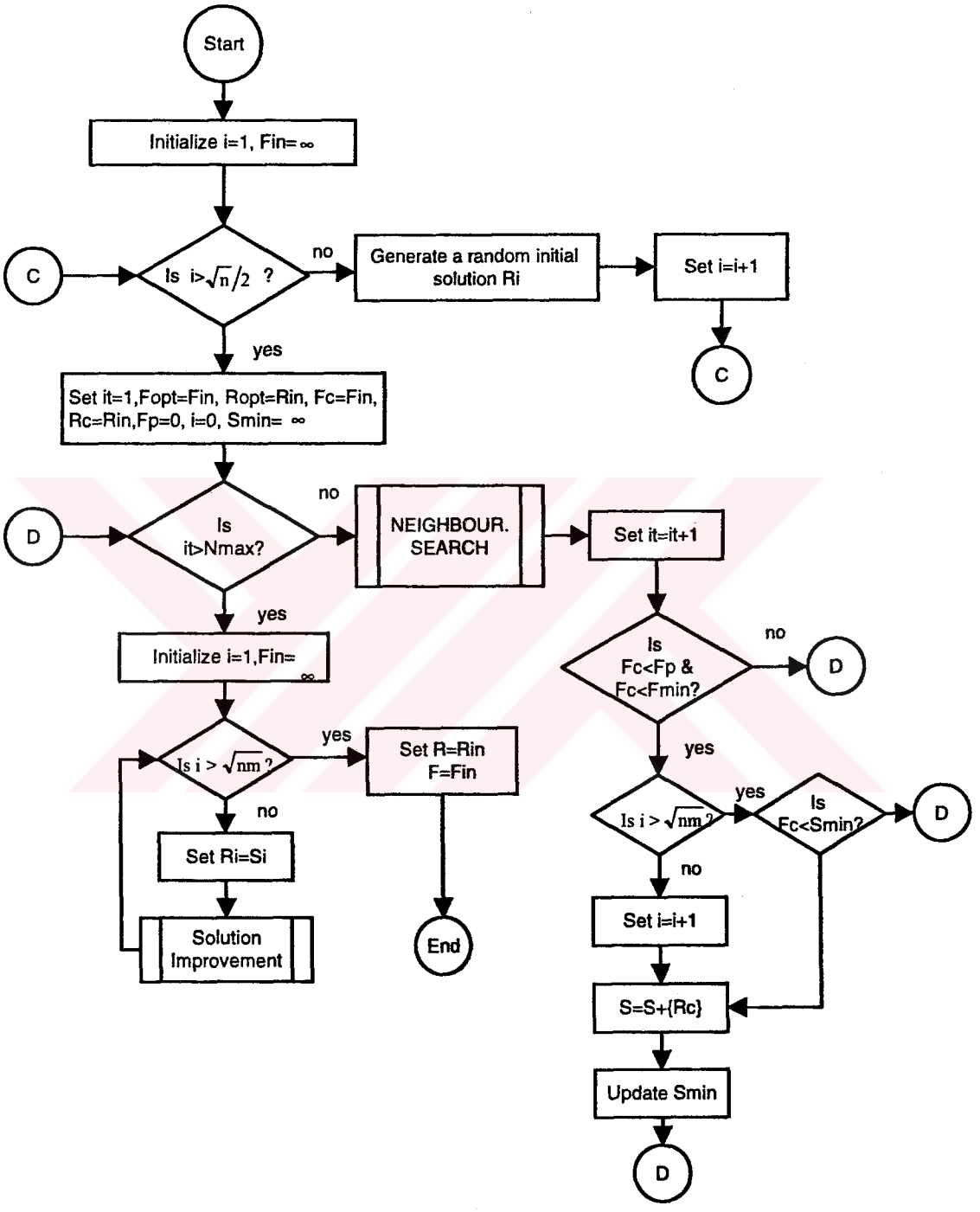

FIGURE 6.2.4. Flow chart of DETABA

6.4. Computational Results

DETABA was tested on the 7 test problems (problems 1-5 and 11-12) given in Christofides,Mingozzi and Toth [59]. These problems contain between 50 and 199 vertices as well as the depot. Without loss of generality, the selected problems have only capacity restrictions, but do not have route length constraints. However, the experiments can be directly extended to the later case. In the problems 1-5, the locations of the vertices are randomly generated over the plane while in the problems 11-12 vertices appear in clusters while Euclidean distances are used in all these cases. The travel time matrix coincides with the distance matrix and there are no service times at the vertices. All computations were performed with full real precision and the solutions were reported as rounded up or down after the second decimal. The results are provided in Table 6.4.1 in terms of the objective function values after the first improvement, solution improvement and intensification phases, and the computational time in CPU(sec). As it can be seen easily, the intensification phase proposed in this study brings considerable improvement especially as the problem size gets larger.

TABLE 6.4.1. Objective function values in each DETABA phase for the test problems

Problem	*n*	*First Improvement Best Solution*	*Solution Improvement Best Solution*	*Intensification Best Solution*	*CPU(sec)*
1	50	810.00	530.75	524.61	695.61
2	75	1344.85	836.71	836.71	3900.93
3	100	1387.47	836.50	836.02	2039.72
4	150	2008.72	1047.62	1043.89	11227.63
5	199	2538.74	1331.48	1325.99	22595.05
11	120	1343.41	1074.90	1072.76	5212.41
12	100	1159.30	823.45	820.23	2277.25

Furthermore, comparisons were made between **DETABA** and the other heuristics for which computational results are already available in literature. It is realized that all classical algorithms are clearly dominated by **DETABA** as it is generally the situation when tabu search algorithms are applied to VRP. The fact that metaheuristics such as simulated annealing and tabu search performs far better in VRP than the classical approximate approaches due to their extensive search process is discussed in Gendreau, Hertz and Laporte [55]. Thus only the solution values for the tabu search algorithms developed by Willard, and Pureza and França which are described in [52], Osman [54], Gendreau, Hertz and Laporte [55], and Taillard [57] are reported in Table 6.4.2. As it can be seen, the best known solutions are always obtained by Taillard's tabu search followed by TABUROUTE. The novel performance of Taillard's algorithm is mainly due to the fact that Taillard first decomposes the main problem into subproblems and carries out a more intensified search over each subproblem. The objective function value for Problem 1 is found to be optimal in Christofides,Hadjiconstantinou and Mingozzi [60], and **DETABA** obtains the same optimal solution C*=524.6. For all the other problems the solutions of **DETABA** are worse than the results obtained by Taillard's algorithm and TABUROUTE; however, in Problems 2-5 the solutions of **DETABA** are better than those of other algorithms which are compatible with **DETABA** in terms of the main characteristics of the neighbourhood construction manner by considering only the feasible moves. In Problems 11-12, the performance of **DETABA** declines since the effect of clustering has a negative effect on the neighborhood search algorithm **TANE.** In the process of determining the vertices to be exchanged between routes in a random manner, the relative distances are not considered in **DETABA.** Since only a given number of neighbours are generated, the random generation of vertices may result in the violation of distance optimization and can not be corrected in a limited number of neighbours. Either by increasing the number of neighbours generated in each iteration or by simply modifying the vertex selection rule and incorporating the distance information, it is possible to improve the performance of **TANE** and consequently **DETABA.** However since this leads to a considerable increase in the computational effort, the general framework of the algorithm which works pretty well in case of random locations with a reasonable CPU time is not altered. In the comparison of run times, one should be very careful about making equitable comparisons. It is observed that the run times are extremely sensitive to parameters and the seeds of random number

generators used in the various procedures of the algorithm. The hardware configuration is of course another very important factor affecting the run time. Since the full details of the above mentioned algorithms are not available and furthermore since the hardwares used in each case possess different capabilities, comparison of run times are not included across the algorithms.

TABLE 6.4.2. Computational comparison of various TS algorithms

Problem	*Willard*	*Pureza& França*	*Osman (BA)*	*Osman (FBA)*	*TABUROUTE*	*Taillard*	*DETABA*
1	588	536	524.61	524.61	524.61	524.61	524.61
2	893	842	844	844	835.77	835.26	836.71
3	906	851	835	838	829.45	826.14	836.02
4	-	1081	1052	1044.35	1036.16	1028.42	1043.89
5	-	-	1354	1334.55	1322.65	1298.79	1325.99
11	-	1049	1042.11	1043	1073.47	1042.11	1072.76
12	-	829	819.59	819.59	819.56	819.56	820.23

7. EXPERIMENTAL DESIGN AND COMPUTATIONAL RESULTS

The experimental design is carried out in a manner that will cover all parameters which might affect the performance of the heuristics described in previous chapters. The algorithms were applied to 120 problems which are categorized into ten data sets. Each data set is uniquely characterized by the number of customers, number of products, number of depots and planning horizon which are randomly generated from U(2,10), U(2,5), U(2,10) and U(4,12), respectively. The main characteristics of data sets are provided in Table 7.1.

TABLE 7.1. Characteristics of data sets.

	Data Set 1	Data Set 2	Data Set 3	Data Set 4	Data Set 5	Data Set 6	Data Set 7	Data Set 8	Data Set 9	Data Set 10
No. of customers	3	2	6	6	9	4	5	8	10	7
No. of products	2	4	3	3	5	5	5	2	4	4
No. of depots	5	8	6	3	9	10	2	4	6	7
Planning horizon	4	4	6	6	5	7	8	9	11	12

In order to assess the interrelated impact of problem attributes on the performance of the algorithms, twelve problems within each data set are designed to differ by the behavior of customer demand, the availability of manufacturing capacity and the relative magnitude of setup and holding costs. Two demand scenarios are developed: Uniform demand (DD) and seasonal demand (dd). In case of uniform demand (DD), d_{it}^{k} is randomly generated from U(3500, 5500). In case of seasonal demand (dd), d_{it}^{k} is determined by

$$d_{it}^{k} = (a_{it}^{k} + b_{it}^{k} t) f_{it}^{k} \tag{7.1}$$

where a_{it}^k is chosen from U(3500, 5500), b_{it}^k is chosen from U(25, 100) and f_{it}^k is chosen from U(0,1). Next the processing times are generated randomly from U(5, 15) in terms of hours. Based on the demand and processing time figures, the capacity available in the manufacturer A_t is randomly generated from a uniform distribution on (AL, AU). Here AL is determined recursively as the minimum capacity required in time t and AU is the upper bound selected to control the tightness of capacity respectively given by

$$AL = \max\left\{ \sum_{r=1}^{t} \sum_{i=1}^{I} \sum_{k=1}^{n} d_{it}^k a_i - \sum_{r=1}^{t-1} A_r, \quad 0 \right\} \tag{7.2}$$

$$AU = \alpha \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{k=1}^{n} d_{it}^k a_i \tag{7.3}$$

Two different capacity schemes are assumed for each problem especially in order to analyze the sensitivity of the Lagrangean heuristic to the capacity availability, and α is a parameter used for this purpose. In case of "relaxed" capacity (CC), α is chosen to be 0.75 while in case of "tight" capacity (cc) α is chosen to be 0.30. Holding costs and unit variable transportation costs are also chosen randomly from uniform distributions on (5, 8) and (50, 80) respectively.

The relative magnitude of setup cost of production and setup cost of transporting material from the manufacturer to the depots is presumed to be another problem attribute that will affect the solution procedure. Since setup costs influence lotsizing directly, different lotsizing policies in production and distribution stages may generate conflicting trends in the solution of the relaxed problem. This will of course affect the communication between production and distribution and, consequently, the whole algorithm. To reveal this

interrelation, three different setup cost structures are designed. First a "randomly uniform" cost structure (ss) is developed by choosing s_{it}, randomly from a uniform distribution on

$$U(\sum_{t=1}^{T}\sum_{k=1}^{n} d_{it}^{k} ho_{it}/T, \sum_{t=1}^{T}\sum_{k=1}^{n} d_{it}^{k} ho_{it}) \tag{7.4}$$

and fo_{ijt} and f_{ijt}^{k} from

$$U(\sum_{t=1}^{T}\sum_{k=1}^{n} d_{it}^{k} h_{ijt}/T, \sum_{t=1}^{T}\sum_{k=1}^{n} d_{it}^{k} h_{ijt}) \tag{7.5}$$

In the second scenario (sS), the distribution setup cost is designed to dominate the production setup cost by enforcing the condition that the ratio of s_{it} / ho_{it} to fo_{ijt} / h_{ijt} be 1:5. In the final scenario (Ss), it is required to maintain a ratio of 5:1 for the same ratio. The combination of these scenarios results in 12 test problems in each data set.

The locations of customers are also randomly generated from U(5,70) in terms of miles, and a scaling factor α of 100,000 per mile is used to convert distances to costs. Customer location is invariate within each data set. For each data set, the same customer locations are preserved in all the 12 scenarios. A general procedure of generating test problems is provided in Figure 7.1.

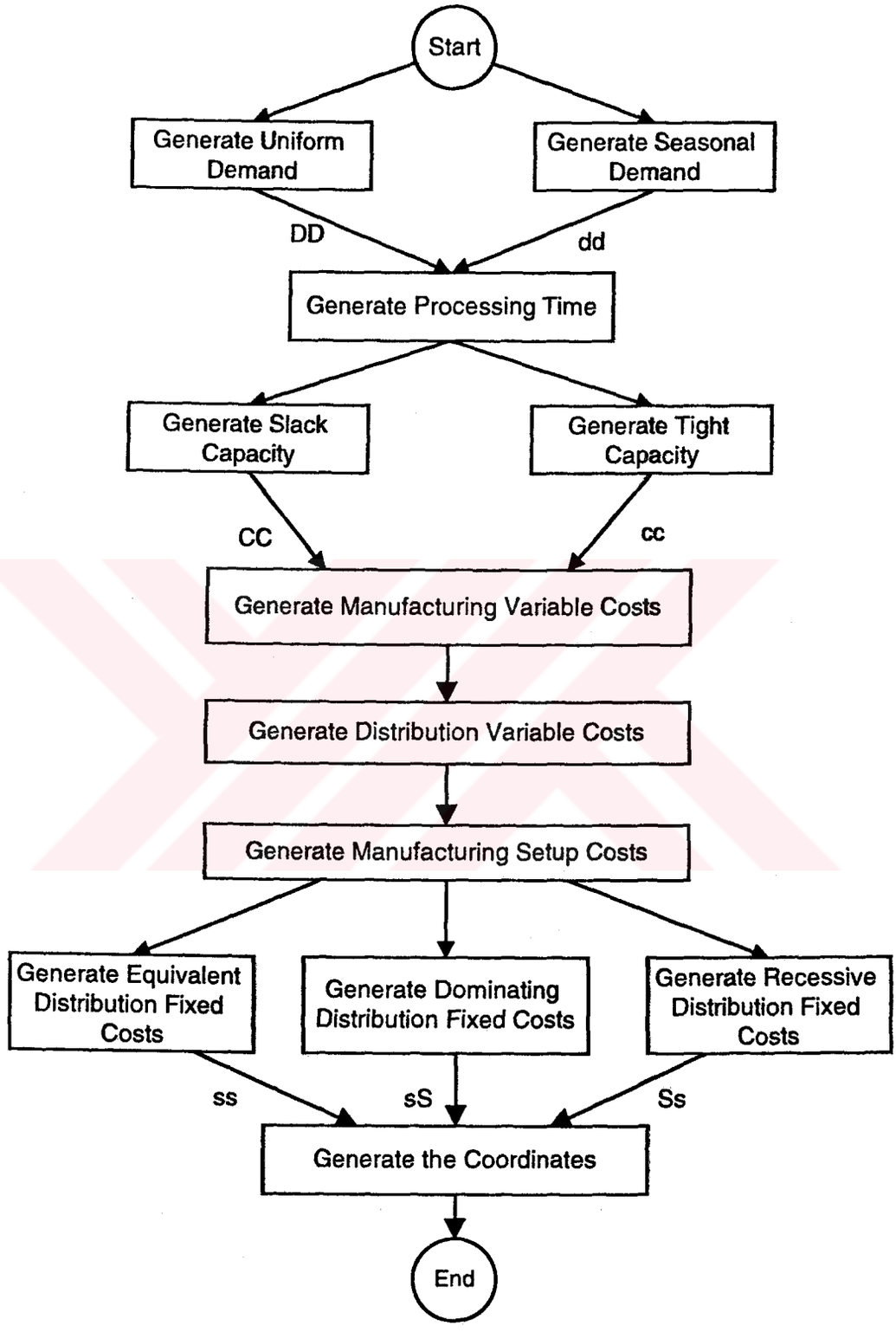

FIGURE 7.1. Test Problem Generation Procedure

First each test problem is solved by using the Lagrangean heuristic developed in this study, and an interface program translated the distribution results into the format required by DETABA. Then the VRP solutions are obtained.

The interface program searches and determines the item-customer pairs which are served by each depot in any time as the depot's customer nodes for that time period. For example, in scenario one of the data set nine the interface program has determined (1,7), (3,1), (3,7), (4,1), (4,4), (4,5), (4,7) and (4,9) as item-customer (i,k) pairs that are served from depot one in time period one. A possible vehicle routing solution of (j,t)=(1,1) is shown in Figure 7.2.

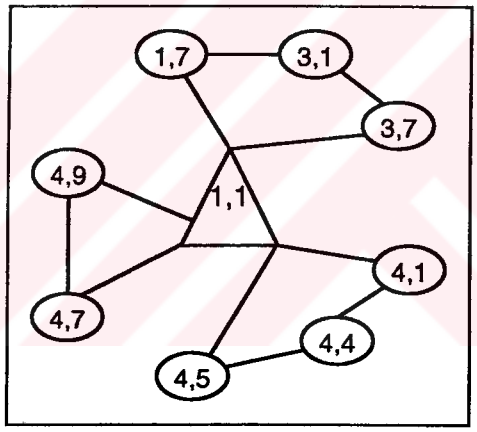

FIGURE 7.2. A possible vehicle routing solution for (j,t)=(1,1)

Here, three vehicles/routes are sent/used to visit customers. One vehicle serves customer seven for item one (1,7), customer one for item three (3,1) and finally customer seven for item three (3,7). The second vehicle serves (4,1), (4,4), (4,5) item-customer pairs. Finally, the last vehicle serves (4,7) and (4,9) item-customer pairs.

First, performance of DISTAL is analyzed by testing it on 30 random distribution problems where there is only one customer demanding one type of product. DISTAL is

coded in C programming language and compiled with Turbo C++ 3.0. GAMS 2.25 solutions are compared with results obtained from our algorithm, and Table A.7.2 summarizes the objective function values and the computation times for 30 random problems with different depot numbers (m) and planning horizon (T).

The Lagrangean relaxation algorithm is coded in C programming language and compiled with Turbo C++ 3.0 on Pentium 133 MHz-32MB Ram. The code of the algorithm is given in Appendix B. The lower and upper bounds and the CPU times for each test problems are obtained. Furthermore it is attempted to solve each problem with the GAMS 2.25 XA solver on the same machine to be compared against the results of the algorithm. All the results are provided in Table A.7.3.

A thorough analysis of the results reveals the following observations:

(a) In 28 problems GAMS can not even find an integer solution although the algorithm finds a solution in a reasonable amount of CPU time;

(b) In 37 problems GAMS solution turns out to be greater than the upper bound obtained by the algorithm;

(c) In 57 problems the algorithm is able to bracket the GAMS solution in a considerably less amount of CPU time;

(d) Only in 13 problems the GAMS solution turns out to be smaller than the lower bound obtained by the algorithm, but the computation time is comparatively enormous.

In short, the algorithm proposed in the study provides good bounds in short time even for the large problems which can not be solved by a well-known commercial code.

When one tries to identify a correlation between the problem parameters and the performance of the algorithm, the following conclusions can be made:

(a) In all the data sets, the smallest difference between the upper and lower bounds occurs in some scenario where capacity is tight (cc). Furthermore the highest difference occurs in some scenario where capacity is relaxed (CC). This implies that the Lagrangean heuristic performs worse as the capacity increases as it is expected; the upper bound cannot be reduced further and the lower bound cannot be increased since production problem in each period ($L1_t$) is minimized as much as the capacity available. The more slack the capacity is, the more ($L1_t$) and consecutively (L) can be minimized.

(b) As far as the CPU time is concerned, the algorithm takes the longest time in the scenario dd/cc/sS in 6 data sets and the second longest time in 3 data sets of the same scenario. This is also intuitive since demand seasonality, tight capacity and sS setup structure constrain the feasible region more.

(c) In the 37 problems mentioned above in which the GAMS solution is worse than the solution of the algorithm, the setup structure is sS where the distribution setup cost is much higher than the manufacturing setup cost. This implies that sS further complicates the GAMS effort to find a feasible integer solution.

After solving 120 test problems by Lagrangean relaxation algorithm, VRP is solved for each of them using DETABA, the prosed vehicle routing algorithm. DETABA is coded in C programming language and compiled with Turbo C++ 3.0 on Pentium 133 MHz-32MB Ram. The code is given in Appendix C. Data set nine is chosen to be given as a sample for VRP results since this is the only data set where GAMS 2.25 XA solver could not find feasible solution for each of its 12 scenarios. Table A.7.4 shows the vehicle routing cost of each (j,t) pair, the total CPU time and the total cost corresponding to 12 scenarios. The results show that VRP can be solved easily and in a reasonable time for each test problem with DETABA.

8. CONCLUSION

In this study, the organizational structure of a production-logistics system is tried to be described within the framework of a single mathematical model from the decision processes and information systems point of view. Hence an integrated production-logistics model is developed to describe the system with centralized decision making. However the integrated model is complex, large-scaled and hence difficult to be solved optimally.

Hierarchical decomposition is applied to the integrated production-logistics system to decentralize the decision process into two submodels: The first submodel tries to optimize the production and distribution functions simultaneously while the second submodel addresses the vehicle routing decisions based on the optimal production and distribution decisions. Hence, a partially decentralized organization is created by hierarchical decomposition.

In the process of solving the first-level aggregate production-distribution model, Lagrangean relaxation approach is used to decouple the imbedded distribution and production decision subproblems. It is realized that both models are further decomposable into smaller problems which can be solved optimally or at least near optimally. An approximation algorithm DISTAL is proposed to solve the two-echelon multi-depot distribution problem using the shortest path principles and shown to solve better or at least the same solution as GAMS XA solver in all, but one case with better CPU time from 30 randomly generated test problems. In fact it is better in 14 cases.

Although the production and distribution subproblems are solved independently in the first level of hierarchical decomposition, the information exchange between them is coordinated by the subgradient optimization procedure. Furthermore the Lagrangean heuristic which aims to resolve the capacity infeasibility and inconsistency between the production and distribution subproblems is monitored by the hierarchical interdependency of the information flow in the system. It is seen that the decision problem placed in the top

level of this hierarchical planning dominates the lower level decisions naturally and influences the performance of the Lagrangean heuristic, the upper bound and consequently the whole Lagrangean relaxation approach. To test the performance of the Lagrangean relaxation algorithm, ten random data sets are generated with 12 different scenarios. Thus, the total 120 problems are solved both with the algorithm and with the GAMS 2.25 XA solver. The results show that the Lagrangean relaxation algorithm proposed in this study provides good bounds in short time even for the large problems which can not be solved by a well-known commercial software.

After solving the aggregate production-distribution problem in level one of the hierarchical decomposition, the solution obtained is used to solve the second level problem which corresponds to the single depot vehicle routing problem. Since customer-product pairs are already assigned to the depots, a decentralized logistics management suffices to solve a single-depot VRP for each depot in each period. VRP is solved for each of the test problems using DETABA, the proposed tabu search vehicle routing algorithm. Data set nine is chosen to be given as a sample for VRP results since this is the only data set where GAMS 2.25 XA solver could not find feasible solution for each of its 12 scenarios. The results show that VRP can be solved easily and in a reasonable time for each test problem with DETABA. A neighbourhood construction heuristic TANE is developed inside DETABA to construct feasible neighbours in each iteration. DETABA performs pretty well in terms of cost and CPU time compared to the other tabu search algorithms in the literature.

The implications of hierarchical decomposition proposed in this study corresponds to a decentralized organizational structure in a company. There are totally three levels of business functions in the whole system which have interactions and feedback among them. The first and second levels correspond to marketing & sales and production planning units in a company respectively. Since JIT philosophy exists over the whole system, distribution decisions pull manufacturing decisions, but when infeasibility occurs, this is provided as a natural feedback from the production planning unit to the marketing & sales unit and the overall optimality is tried to be reached interactively. The third level which is the vehicle routing problem corresponds to the logistics unit in a company. The vehicle routing

problem is considered in the first and second levels as an input to the cost function of the aggregate production-distribution problem. So there is a hidden feedback from the logistics unit to the upper units. Marketing & sales and production planning give their integrated decisions to the logistics unit as an input to its own problem. Feedback from the logistics unit to the upper level units is provided when necessary. By this way conflicts among critical departments in a company are tried to be minimized with the feedbacks provided in the hierarchical structure.

When the study is analyzed as a whole, it can be seen that the integrated production-logistics problem can be solved efficiently with the proposed approach and solution methods. Hence the decomposition style proposed in this study can be clearly implemented in the organizational design of a company.

Suggestions for further study:

The information link from the lower level to the upper level in the hierarchical decomposition has not been studied extensively in this study. To establish a robust hierarchical structure between the aggregate production-distribution submodel and the vehicle routing submodel, this information link should be strengthened.

In this study, it is assumed that the depots are regional dealers each of which solves its own vehicle routing problem. Hence a single depot VRP algorithm is developed. Another case can be such that all depots belong to the manufacturer and it is possible to traverse between the depots. This case corresponds to multi-depot VRP in the literature. So, the lower level problem in the hierarchical decomposition can be solved as a multi-depot VRP.

In this study, it is assumed that vehicle routing lead times are negligible compared to the time-bucket, so they are not included in the formulations and solution procedures. The VRP with time windows can be studied in the lower level of the hierarchical decomposition as a further study.

It is seen in the computational study that when the customer vertices appear in clusters, the performance of DETABA declines since clustering has a negative effect on the neighbourhood search algorithm TANE. Hence TANE and DETABA can be improved as a further study.

It is assumed in this study that there is no limitation upon stock holding capability of the distribution centers. Thus, an algorithm can be developed for the distribution problem considering this constraint as a further study.

APPENDIX A

THE RESULTS OF COMPUTATIONAL STUDY

TABLE A.6.4.3. Best VRP solution results for the test problem 1

Problem # 1: n=50, Q=8000		*Total Duration:*	524.61
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 38 9 30 34 50 16 21 29 2 11 0	7950	99.33
2	0 47 4 17 42 19 40 41 13 18 0	7850	109.06
3	0 32 1 22 20 35 36 3 28 31 26 8 0	7450	118.52
4	0 46 5 49 10 39 33 45 15 44 37 12 0	8000	99.25
5	0 6 14 25 24 43 7 23 48 27 0	7600	98.45

TABLE A.6.4.4. Best VRP solution results for the test problem 2

Problem # 2: n=75, Q=7000		*Total Duration:*	836.71
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 35 11 66 65 38 58 0	6900	81.81
2	0 62 22 64 42 41 43 1 73 51 0	6900	100.01
3	0 17 3 44 32 9 39 72 12 0	6900	65.62
4	0 16 49 24 56 23 63 33 6 0	7000	92.69
5	0 30 74 21 61 28 2 68 0	7000	74.38
6	0 46 52 27 13 57 15 5 45 0	6950	73.55
7	0 7 53 14 59 19 54 8 0	6950	95.33
8	0 29 37 20 70 60 71 69 36 47 48 0	6750	103.30
9	0 10 31 25 55 18 50 40 0	7000	117.45
10	0 26 67 34 4 75 0	5850	32.58

TABLE A.6.4.5. Best VRP solution results for the test problem 3

Problem # 3: n=100, Q=10000		*Total Duration:*	836.02
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 73 41 23 67 39 25 55 24 29 68 80 12 28 0	9800	130.06
2	0 96 99 59 93 85 61 5 84 17 45 46 8 83 60 0	9950	104.70
3	0 27 31 88 62 10 63 90 32 51 9 81 33 50 0	9950	111.47
4	0 53 40 21 72 74 22 75 56 4 54 26 0	8150	77.98
5	0 13 87 97 95 94 6 0	5550	43.21
6	0 92 98 37 100 91 16 86 38 44 14 42 43 15 57 2 58 0	4900	126.66
7	0 52 7 19 11 64 49 36 47 48 82 18 89 0	4650	118.55
8	0 76 77 3 79 78 34 35 71 65 66 20 30 70 1 69 0	9950	123.39

TABLE A.6.4.6. Best VRP solution results for the test problem 4

Problem # 4: n=150, Q=200		*Total Duration:*	1043.89
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 144 57 15 43 38 140 86 113 17 84 5 118 60 0	198	114.84
2	0 53 58 2 115 145 41 22 133 75 74 72 73 21 105 0	197	70.74
3	0 40 110 4 56 23 67 39 139 25 55 130 149 26 0	200	110.25
4	0 88 148 62 123 19 107 11 64 49 143 36 47 7 0	196	120.69
5	0 3 78 34 135 35 136 65 71 66 103 9 120 51 122 0	199	134.63
6	0 54 134 24 29 121 129 79 81 33 102 50 1 132 0	195	89.17
7	0 112 94 13 0	79	27.72
8	0 89 147 6 96 104 99 93 85 98 92 59 95 0	195	53.72
9	0 117 97 37 100 91 61 16 141 44 119 14 142 42 87 137 0	200	86.46
10	0 18 83 114 8 125 45 46 124 48 82 106 52 146 0	189	89.95
11	0 69 101 70 30 20 128 131 32 90 63 126 108 10 31 127 0	197	89.89
12	0 27 111 76 116 77 68 80 150 109 12 138 28 0	190	55.82

TABLE A.6.4.7. Best VRP solution results for the test problem 5

Problem # 5: n=199, Q=200		*Total Duration:*	1325.99
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 176 50 102 185 79 158 3 77 196 184 28 0	199	57.07
2	0 40 73 171 133 75 23 186 56 197 72 180 0	200	76.93
3	0 198 110 155 4 139 187 39 67 170 25 55 165 179 0	200	96.12
4	0 112 13 137 87 144 57 178 115 2 152 58 0	200	54.07
5	0 138 154 12 150 80 177 109 195 26 105 53 156 0	200	51.65
6	0 94 59 98 85 93 104 99 96 6 147 89 0	200	50.94
7	0 122 128 20 188 66 65 136 35 135 164 34 78 169 129 0	200	122.53
8	0 60 118 5 84 173 61 113 17 45 125 199 83 166 0	200	83.36
9	0 167 190 88 182 48 82 194 106 153 52 146 0	199	65.77
10	0 111 76 116 68 121 29 24 163 134 54 130 149 0	196	77.65
11	0 117 172 42 142 14 119 44 192 91 193 100 37 151 95 183 0	200	79.57
12	0 157 33 81 120 9 161 71 103 51 1 132 0	196	84.35
13	0 21 74 22 41 145 15 43 38 140 86 16 141 191 92 97 0	200	128.35
14	0 70 90 126 63 64 49 143 36 47 168 124 46 174 8 114 18 0	200	130.93
15	0 27 69 162 108 189 10 159 62 148 31 127 0	200	64.97
16	0 101 30 160 131 32 181 11 175 107 19 123 7 0	196	101.73

TABLE A.6.4.8. Best VRP solution results for the test problem 11

Problem # 11: n=120, Q=200		*Total Duration:*	1072.76
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 82 111 86 112 84 117 113 83 6 5 4 3 1 2 81 119 120 0	187	120.83
2	0 91 90 114 18 118 108 8 12 13 14 15 11 10 9 7 85 88 0	198	119.97
3	0 17 16 19 25 22 24 27 33 30 31 34 36 29 35 32 28 26 23 20 21 109 0	197	207.94
4	0 95 37 38 39 42 41 44 47 46 49 50 51 48 45 43 40 0	200	199.63
5	0 105 53 55 58 56 60 63 66 64 62 61 65 59 57 54 52 0	200	214.22
6	0 107 67 69 70 71 74 75 72 78 80 79 77 76 73 68 106 0	199	144.55
7	0 87 89 92 93 96 94 97 115 110 98 116 100 103 104 99 101 102 0	194	65.62

TABLE A.6.4.9. Best VRP solution results for the test problem 12

Problem #12: n=100, Q=200		*Total Duration:*	820.23
Route #	Sequence of Vertices	Route Load	Route Duration
1	0 34 36 39 38 37 35 31 33 32 0	200	97.23
2	0 57 59 60 58 56 53 54 55 0	200	101.88
3	0 43 41 40 42 44 45 46 48 51 50 52 49 47 0	160	65.48
4	0 81 78 76 71 70 73 77 79 80 72 61 64 68 69 0	200	137.02
5	0 90 87 86 83 82 84 85 88 89 91 0	170	76.07
6	0 98 96 95 94 92 93 97 100 99 0	190	95.94
7	0 13 17 18 19 15 16 14 12 10 0	200	96.04
8	0 75 1 2 4 6 9 11 8 7 3 5 0	170	56.17
9	0 20 24 25 27 29 30 28 26 23 22 21 0	170	50.80
10	0 66 62 74 63 65 67 0	150	43.59

TABLE A.7.2. Computational results of DISTAL

					GAMS		Algorithm	
Prob. #	m	T	# binary var.s	# real var.s	CPU (sec)*	Best Solution	CPU (sec)*	Best Solution
1	4	3	24	36	0.70	14707.95	0.055	14707.95
2	4	4	32	48	0.95	17016.60	0.055	17016.60
3	4	5	40	60	0.83	17328.81	0.055	17328.81
4	4	6	48	72	1.10	22935.65	0.055	22701.35
5	4	7	56	84	1.16	37264.44	0.055	37264.44
6	4	8	64	96	1.14	41362.10	0.055	41359.05
7	4	9	72	108	1.20	53894.47	0.055	53894.47
8	4	10	80	120	1.20	46852.48	0.055	46852.48
9	4	11	88	132	1.31	83067.77	0.055	82742.10
10	4	12	96	144	1.37	62336.95	0.055	62113.20
11	5	6	60	90	0.94	31397.82	0.055	31397.82
12	5	7	70	105	1.27	23627.23	0.055	23627.23
13	6	7	84	126	1.16	36120.80	0.055	36120.80
14	6	7	84	126	4.07	8644.44	0.055	8644.44
15	6	8	96	144	1.22	29316.50	0.110	29280.40
16	6	8	96	144	3.97	9082.74	0.055	9098.84
17	6	9	108	162	1.20	32198.50	0.055	31821.70
18	6	9	108	162	6.60	9689.08	0.110	9689.08
19	6	12	144	216	1.48	70111.63	0.110	69946.19
20	7	7	98	147	1.16	38261.20	0.110	38244.70
21	7	8	112	168	1.27	39003.50	0.110	38990.00
22	7	8	168	224	1.32	68735.80	0.110	68735.80
23	7	9	126	189	1.37	38953.50	0.055	38953.50
24	7	9	126	189	1.44	65446.60	0.110	65437.00
25	7	12	168	252	1.75	62567.35	0.110	62567.34
26	8	7	112	168	6.33	14204.90	0.165	14174.54
27	8	8	128	192	1.47	10766.18	0.055	10597.68
28	8	8	128	192	1.44	35852.4	0.055	35757.40
29	8	9	216	288	1.71	14523.14	0.110	14523.14
30	8	10	160	240	1.65	48089.20	0.110	48073.20

* 80486 DX2-66

TABLE A.7.3. Computational results of the Lagrangean relaxation algorithm

Data Set	Scenario	Algorithm				GAMS	
		Zlower	Zupper	Diff(%)	CPU(sec)	Best solution	CPU(sec)
1	DD/cc/ss	2,764,549	2,878,634	3.96	27.14	2,746,121	12.24
	dd/cc/ss	1,638,702	1,651,390	0.77	12.09	1,475,800	16.30
	DD/CC/ss	2,676,019	2,833,886	5.57	26.87	2,746,121	12.96
	dd/CC/ss	1,523,510	1,621,508	6.04	27.09	1,475,800	14.31
	DD/cc/sS	202,337,408	204,202,896	0.91	15.82	206,112,973	12,925.05
	dd/cc/sS	115,491,168	116,826,136	1.14	32.31	118,494,297	9,950.11
	DD/CC/sS	199,249,792	201,879,472	1.30	27.91	206,141,014	16,609.50
	dd/CC/sS	111,177,424	112,116,360	0.84	11.81	118,494,297	9,459.19
	DD/cc/Ss	187,382,560	188,582,384	0.64	24.34	186,151,299	40.97
	dd/cc/Ss	102,607,160	103,561,120	0.92	13.13	99,490,969	1,411.86
	DD/CC/Ss	182,181,552	187,988,752	3.09	27.25	186,151,299	218.44
	dd/CC/Ss	99,081,704	100,506,856	1.42	27.20	99,490,969	1,411.30
2	DD/cc/ss	3,481,961	3,739,790	6.89	43.57	3,598,875	13.94
	dd/cc/ss	2,015,814	2,137,519	5.69	44.40	2,080,917	618.24
	DD/CC/ss	3,447,146	3,657,446	5.75	42.64	3,598,875	14.89
	dd/CC/ss	2,015,814	2,137,519	5.69	43.30	2,080,917	606.98
	DD/cc/sS	264,052,336	274,026,400	3.64	46.21	284,238,063	23,448.78
	dd/cc/sS	163,495,568	168,649,280	3.06	51.54	185,485,106	19,237.08
	DD/CC/sS	258,868,000	268,085,968	3.44	44.56	284,159,796	24,639.00
	dd/CC/sS	159,107,072	164,282,496	3.15	44.56	185,485,106	18,861.29
	DD/cc/Ss	241,462,880	251,141,088	3.85	45.66	245,499,392	9,595.64
	dd/cc/Ss	143,638,672	148,638,912	3.36	50.77	145,221,762	18,878.53
	DD/CC/Ss	239,547,456	249,072,880	3.82	44.73	245,499,392	9,563.34
	dd/CC/Ss	139,854,032	146,026,096	4.23	43.24	145,199,491	17,627.50
3	DD/cc/ss	850,788,096	870,490,880	2.26	126.76	836,548,611	20,002.05
	dd/cc/ss	460,244,928	477,380,448	3.59	113.30	467,845,630	20,002.37
	DD/CC/ss	814,925,312	849,030,464	4.02	108.24	850,241,990	20,002.27
	dd/CC/ss	468,282,720	486,477,536	3.74	122.80	486,074,679	20,002.20
	DD/cc/sS	916,351,104	938,949,440	2.41	131.32	993,257,363	76,454.28
	dd/cc/sS	480,465,376	495,537,664	3.04	116.76	564,647,518	10,683.75
	DD/CC/sS	885,951,104	912,172,480	2.87	128.30	984,059,707	42,262.64
	dd/CC/sS	508,413,472	531,011,168	4.26	110.44	592,586,791	36,961.33
	DD/cc/Ss	831,677,696	851,370,240	2.31	124.95	834,011,299	20,011.75
	dd/cc/Ss	453,279,840	469,908,224	3.54	112.64	451,040,032	20,012.74
	DD/CC/Ss	850,732,672	870,759,360	2.30	113.90	838,219,739	20,011.37
	dd/CC/Ss	430,126,848	450,843,872	4.60	116.32	452,082,941	20,011.38
4	DD/cc/ss	855,918,656	881,509,184	2.90	79.62	857,707,955	25,001.64
	dd/cc/ss	489,687,872	500,178,560	2.10	82.53	494,732,654	25,001.08
	DD/CC/ss	863,883,840	913,410,176	5.42	79.73	898,857,045	25,002.02
	dd/CC/ss	471,623,456	489,061,632	3.57	77.03	484,754,239	25,001.79
	DD/cc/sS	961,982,464	986,922,240	2.53	88.90	1,014,041,319	20,001.40
	dd/cc/sS	487,873,440	497,948,832	2.02	115.66	520,423,373	19,530.72
	DD/CC/sS	947,745,728	972,913,920	2.59	77.03	1,013,517,352	20,001.60
	dd/CC/sS	564,363,456	583,559,424	3.29	89.95	603,188,218	20,001.50
	DD/cc/Ss	843,721,920	866,348,864	2.61	78.24	840,769,968	20,001.77
	dd/cc/Ss	418,949,760	429,877,408	2.54	78.68	420,483,102	20,001.60
	DD/CC/Ss	827,530,176	861,880,448	3.99	74.51	846,298,176	20,001.56
	dd/CC/Ss	486,909,600	505,163,040	3.61	74.29	501,924,330	20,001.71
5	DD/cc/ss	1,734,153,472	1,808,592,640	4.12	534.23	1,753,385,194	20,038.75
	dd/cc/ss	916,100,032	937,298,560	2.26	527.36	-	20,059.28
	DD/CC/ss	1,758,256,256	1,805,869,440	2.64	527.80	1,789,108,393	20,044.74
	dd/CC/ss	901,170,688	932,817,344	3.39	531.54	-	20,056.64
	DD/cc/sS	1,895,507,200	1,956,205,568	3.10	569.23	2,139,617,242	20,011.20
	dd/cc/sS	1,065,204,992	1,102,081,024	3.35	803.46	1,293,096,051	20,011.42
	DD/CC/sS	1,790,151,424	1,872,313,600	4.39	676.70	2,118,291,153	20,008.99
	dd/CC/sS	1,017,123,072	1,072,288,768	5.14	669.78	1,242,943,281	20,008.95
	DD/cc/Ss	1,695,042,688	1,767,304,960	4.09	670.82	1,745,086,665	20,008.90
	dd/cc/Ss	903,904,832	923,992,960	2.17	545.16	900,472,994	20,008.95
	DD/CC/Ss	1,651,161,216	1,733,139,072	4.73	550.38	1,713,347,029	20,009.11
	dd/CC/Ss	863,369,536	898,320,832	3.89	528.74	894,939,694	20,017.17

TABLE A.7.3. Computational results of the Lagrangean relaxation algorithm (continued)

Data Set	Scenario	Algorithm				GAMS	
		Zlower	Zupper	Diff(%)	CPU(sec)	Best solution	CPU(sec)
6	DD/cc/ss	1,059,230,208	1,104,600,576	4.11	241.21	1,095,696,648	20,025.65
	dd/cc/ss	583,752,128	610,881,600	4.44	244.29	624,205,688	20,027.73
	DD/CC/ss	1,060,018,048	1,130,596,864	6.24	238.52	1,122,215,715	20,028.05
	dd/CC/ss	548,115,712	596,088,896	8.05	230.44	610,567,155	20,027.29
	DD/cc/sS	1,172,623,232	1,225,459,712	4.31	261.81	-	20,035.98
	dd/cc/sS	665,944,128	696,872,704	4.44	302.58	-	20,029.83
	DD/CC/sS	1,160,306,816	1,225,993,984	5.36	261.87	-	20,033.34
	dd/CC/sS	638,876,928	666,958,208	4.21	275.55	-	20,033.23
	DD/cc/Ss	1,021,985,216	1,067,523,136	4.27	272.53	1,037,820,816	20,027.49
	dd/cc/Ss	561,256,256	593,805,952	5.48	260.00	598,100,009	20,028.90
	DD/CC/Ss	1,004,304,448	1,072,395,200	6.35	258.13	1,053,700,024	20,030.00
	dd/CC/Ss	545,736,960	583,053,376	6.40	293.02	584,622,523	20,031.04
7	DD/cc/ss	1,685,151,616	1,744,481,536	3.40	127.42	1,649,950,029	20,000.83
	dd/cc/ss	863,540,736	896,964,288	3.73	113.24	887,566,874	20,001.45
	DD/CC/ss	1,612,134,272	1,702,364,800	5.30	106.37	1,664,555,149	20,001.18
	dd/CC/ss	876,660,800	938,290,560	6.57	101.98	928,815,266	20,001.38
	DD/cc/sS	1,746,541,568	1,816,353,664	3.84	111.98	1,860,197,249	20,001.49
	dd/cc/sS	1,045,493,632	1,102,717,056	5.19	119.45	1,117,909,150	20,002.36
	DD/CC/sS	1,756,489,856	1,848,242,560	4.96	106.76	1,889,703,855	20,002.06
	dd/CC/sS	988,746,368	1,036,214,144	4.58	108.35	1,080,284,704	20,001.34
	DD/cc/Ss	1,514,268,288	1,589,512,832	4.73	105.22	1,544,060,812	20,001.05
	dd/cc/Ss	874,201,408	910,860,992	4.02	106.54	892,513,915	20,001.21
	DD/CC/Ss	1,482,067,840	1,622,274,432	8.64	101.98	1,551,041,889	20,001.11
	dd/CC/Ss	901,748,608	967,729,664	6.82	104.18	934,505,374	20,001.16
8	DD/cc/ss	1,217,148,544	1,229,446,912	1.00	135.05	1,171,228,469	20,001.11
	dd/cc/ss	615,857,600	629,586,944	2.18	121.76	645,638,504	20,001.12
	DD/CC/ss	1,177,860,224	1,233,812,736	4.53	110.27	1,237,148,374	20,000.95
	dd/CC/ss	647,536,640	674,986,624	4.07	110.33	693,003,042	20,001.10
	DD/cc/sS	1,264,113,408	1,285,845,504	1.69	123.74	1,396,544,408	18,847.95
	dd/cc/sS	751,150,336	766,690,688	2.03	148.41	866,371,707	20,001.06
	DD/CC/sS	1,217,346,176	1,269,886,336	4.14	112.91	1,378,763,119	20,001.22
	dd/CC/sS	683,218,304	702,086,912	2.69	113.74	796,557,666	20,001.22
	DD/cc/Ss	1,086,666,496	1,113,601,152	2.42	110.00	1,094,514,876	20,000.88
	dd/cc/Ss	613,109,504	624,263,168	1.79	111.70	621,081,094	20,001.00
	DD/CC/Ss	1,164,404,608	1,193,152,896	2.41	115.27	1,136,721,738	20,001.00
	dd/CC/Ss	588,403,328	616,938,816	4.63	106.92	609,772,999	20,001.20
9	DD/cc/ss	3,434,323,968	3,579,186,944	4.05	746.98	-	20,119.10
	dd/cc/ss	1,810,199,040	1,892,682,240	4.36	734.89	-	20,063.12
	DD/CC/ss	3,284,516,864	3,512,175,616	6.48	703.96	-	20,082.90
	dd/CC/ss	1,836,913,920	1,957,291,648	6.15	705.27	-	20,104.27
	DD/cc/sS	3,731,201,024	3,883,367,168	3.92	731.32	-	20,080.97
	dd/cc/sS	2,299,870,976	2,404,287,232	4.34	977.47	-	20,066.46
	DD/CC/sS	3,771,917,568	3,927,322,112	3.96	1,064.95	-	20,145.54
	dd/CC/sS	2,293,321,472	2,404,916,992	4.64	929.89	-	20,151.26
	DD/cc/Ss	3,246,604,800	3,426,278,912	5.24	832.03	-	20,118.15
	dd/cc/Ss	1,912,310,272	2,016,355,456	5.16	717.86	-	20,099.17
	DD/CC/Ss	3,213,678,080	3,455,689,216	7.00	702.31	-	20,057.31
	dd/CC/Ss	1,873,230,720	1,980,615,168	5.42	687.36	-	20,061.19
10	DD/cc/ss	2,628,622,080	2,766,631,424	4.99	565.27	-	20,046.80
	dd/cc/ss	1,442,500,096	1,505,623,424	4.19	666.65	-	20,060.59
	DD/CC/ss	2,593,873,152	2,724,868,864	4.81	546.59	-	20,057.84
	dd/CC/ss	1,501,342,976	1,612,278,400	6.88	644.73	-	20,090.99
	DD/cc/sS	3,014,396,160	3,143,171,328	4.10	1,015.22	-	20,096.46
	dd/cc/sS	1,746,000,896	1,825,797,120	4.37	737.09	-	20,082.16
	DD/CC/sS	2,852,812,032	3,015,247,616	5.39	637.36	-	20,077.62
	dd/CC/sS	1,690,135,424	1,797,989,376	6.00	541.87	-	20,005.77
	DD/cc/Ss	2,528,715,520	2,635,295,744	4.04	670.71	2,549,619,415	20,008.49
	dd/cc/Ss	1,329,581,184	1,404,229,760	5.32	528.63	1,393,866,779	20,010.32
	DD/CC/Ss	2,481,051,392	2,621,626,112	5.36	539.84	-	20,080.53
	dd/CC/Ss	1,373,873,280	1,487,091,968	7.61	530.33	-	20,054.19

TABLE A.7.4. VRP solutions of data set 9

F(j,t)	Problem #1	Problem #2	Problem #3	Problem #4	Problem #5	Problem #6	Problem #7	Problem #8	Problem #9	Problem #10	Problem #11	Problem #12
F(1,1)	21522848	15505394	22445640	12262529	15733897	15323368	13356680	15942132	7268908.5	15637844	9028422	17904694
F(1,2)	12010882	14392549	21900394	10925017	21017602	10996141	18012660	12118032	7202777	15744303	13552538	17492758
F(1,3)	17260360	18276252	15826048	16020734	14502535	11766426	21183504	15655897	19656566	13897433	17142776	15490456
F(1,4)	15717814	15993898	21904320	20373932	18421300	13240801	20076284	15129326	19050396	19158288	16671901	9645933
F(1,5)	21318194	10567249	20281158	16842454	21091424	9801618	9282717	12894062	17696438	18613302	22310520	13627608
F(1,6)	14304981	12239846	11202495	13208306	8925110	12337164	18357084	7717546	19667470	15186330	19939676	9714120
F(1,7)	13741821	16507126	9017610	17509054	15763290	8741208	20601732	10311336	11316892	18238668	12587941	14397900
F(1,8)	24019180	13214688	21617974	13522803	18573442	9801626	11769220	10469004	21813166	15218784	22682416	15400508
F(1,9)	22394690	13876785	15991070	16988698	12411916	13091403	7050014	10706147	24538758	17927714	9017610	13853594
F(1,10)	24286526	11644428	19920436	18987030	16286460	16311847	13283432	14953302	13518324	11011866	20835984	6802946
F(1,11)	15464845	14132128	15375770	16341934	12044258	18074530	8920711	16818868	15880800	8676397	21544328	14019280
F(2,1)	13796916	15745564	15828018	16679412	15827724	17641088	23378446	15435934	18730690	18387756	10249256	18874172
F(2,2)	16704322	19534808	18402434	19635410	18128976	16833948	17135400	17150898	17444336	24002426	14970668	17117630
F(2,3)	13078029	16510840	22634914	20341998	14551258	15187435	19450092	15199050	23057720	17831936	17588426	15854246
F(2,4)	11004770	17534788	12483036	15958800	8531679	11753442	15187430	19484050	20739132	19345162	16132988	15042574
F(2,5)	20225578	11135859	16971350	17662852	15081216	17898246	26017838	14779018	16413050	18902656	18410390	16500044
F(2,6)	17970492	15213297	23493754	22833434	20265018	13565207	22036152	18734516	14690732	17559716	14054290	18925052
F(2,7)	27446940	8950617	16552264	15946956	16452717	20542628	19170596	13472798	25327712	19991206	14562505	15848182
F(2,8)	17575024	17442176	18423548	15082868	17592810	16887294	18854674	14909590	19832508	13871861	20883896	22195404
F(2,9)	18931682	15101644	10061130	21926078	24469020	14682834	20643162	15715036	17339104	14368955	17028338	11945852
F(2,10)	17266480	17743234	30748256	12228911	23863656	16438243	26976554	18034146	17595594	12148130	14504955	17758268
F(2,11)	13353043	15603306	13971399	12385373	20486882	14929186	25311430	14216785	15062536	16979742	22240234	23040616

TABLE A.7.4. VRP solutions of data set 9 (continued)

F(j,t)	Problem #1	Problem #2	Problem #3	Problem #4	Problem #5	Problem #6	Problem #7	Problem #8	Problem #9	Problem #10	Problem #11	Problem #12
F(3,1)	19266760	20839080	22863454	20237720	13222652	21321848	20619808	20230364	33175280	14025510	24664096	14390737
F(3,2)	16785234	23642878	30311008	25042960	24459068	22387850	23413360	19600366	23271874	22404686	31933440	15252819
F(3,3)	15094167	21790286	26250744	18828876	22337788	22872722	24500684	17428720	27677308	17402360	28013080	20791992
F(3,4)	25830334	18924930	26250724	20298736	25321762	22235152	25000964	16321817	21013094	22211292	30982302	21516238
F(3,5)	26138172	22582300	29497444	21167162	17342300	20378152	26685954	21716004	29968080	23185024	27093814	23566408
F(3,6)	33903652	22221270	26640392	10119289	23732374	10236857	29177124	19798908	20595018	17542562	26263940	18489762
F(3,7)	32970000	19230528	31661140	15127826	17577972	14444217	37206280	17228600	26313680	17516344	23564940	24034106
F(3,8)	24587576	21061500	27759092	24187984	14782480	18672232	36526712	20458128	13891458	27437582	17566332	21500964
F(3,9)	33294302	21777080	17400708	19046094	25137090	18869768	27958048	21013084	24114692	21110566	26608316	18438528
F(3,10)	32396322	24158962	24208620	20367916	23735100	15160935	13579348	20997212	25248096	21113238	29425204	19848052
F(3,11)	23248558	19826140	29590946	20604588	14771278	20653230	39601236	15825647	45358224	21727692	20277934	20320442
F(4,1)	16883062	12204580	14683934	19249748	24698100	14098964	21607656	11794034	24645356	17657588	22152514	12578840
F(4,2)	22874224	17504504	19211600	12455542	14856524	20775448	13454470	14335299	20975390	16099010	14205419	21856328
F(4,3)	20726192	15197372	16280880	20060132	17220904	19285870	9857832	12771634	18470090	17050630	16224758	14138824
F(4,4)	14004344	16665906	19997762	15638664	17010150	12151344	8393805	9057328	18660420	9368702	22925102	17191622
F(4,5)	12331088	19127780	13519726	11197599	21262984	12014814	8904252	14394056	17157768	13686099	15953476	8786119
F(4,6)	12847514	15094338	18243048	17893340	16700850	13466021	15254764	12451930	27630440	13697906	19670904	11678302
F(4,7)	21053688	19081552	20051314	17558948	16876254	17424076	11001993	12011208	21208884	11684998	25367900	13103644
F(4,8)	21183038	15156684	16259542	14072112	19720806	17150432	18223680	14856518	16641205	16400124	16368966	13228120
F(4,9)	17806464	17441416	19698778	13239034	17399560	11510529	18005468	18394904	18415136	17815856	28195368	18796004
F(4,10)	3006659.25	15266317	12393649	14789495	15527518	8676682	12085374	17269092	14928467	19820330	15408462	18111762
F(4,11)	23132768	20926152	22060614	18829562	21809828	15456084	12140194	17601730	15650302	10129044	15667183	19090716

TABLE A.7.4. VRP solutions of data set 9 (continued)

F(j,t)	Problem #1	Problem #2	Problem #3	Problem #4	Problem #5	Problem #6	Problem #7	Problem #8	Problem #9	Problem #10	Problem #11	Problem #12
F(5,1)	14293900	13458419	20374576	5099673	19655356	17075172	16427874	14031142	11709682	15572882	24839200	14385474
F(5,2)	20211906	13337464	14712549	10894178	22809076	12085133	18666264	17923656	20912160	12044472	22925054	15861041
F(5,3)	24665352	15112256	7385704	5491832	21844300	12085132	21433292	14638755	12201709	19432764	17255470	17088180
F(5,4)	22526934	18312502	15740262	12537798	13230048	14087683	13962111	14791414	24858676	15416459	15291660	17464404
F(5,5)	11584490	18064272	20718348	15898764	10894181	15270138	16788836	9591547	18116948	12560789	12978106	17931350
F(5,6)	7965348	21858156	17838046	12336244	19472780	19422854	14798706	16978100	17163032	19770084	12724357	18234886
F(5,7)	11313170	16047274	17732920	9954976	10944484	15191522	13737997	23208316	15944682	14693927	19803232	14847958
F(5,8)	10758594	18784486	23190340	14180718	21036316	7719095	14679334	17861088	16941636	21364684	18124692	16846398
F(5,9)	11071353	15286126	30422196	15984804	12219311	16724326	12135439	17284518	15285744	6897215	19864952	11542942
F(5,10)	8957883	10923113	14224382	6189198	15804300	17124400	23444594	8939506	23694092	13386848	13243642	16435165
F(5,11)	16675243	13523111	20085848	10455626	17839748	17231534	18581260	17727682	9955160	10693422	9724886	9888783
F(6,1)	21150584	14084516	12530324	17549366	15673342	14264878	11755650	18176790	21883350	13451018	21301572	13761072
F(6,2)	24393852	15985066	11583707	19937174	12848564	17878040	19544904	14572156	18323780	13494286	15661125	12763630
F(6,3)	18323800	14229648	21599450	13271390	20590132	18624868	17597508	18278238	13134697	5588334.5	21995738	15660818
F(6,4)	7973616	12668368	22370954	13817054	22092554	16458668	21617500	20096606	10697827	5357610.5	16579609	22032492
F(6,5)	5664677	8277905	13499638	22418454	27202524	17478976	16542457	22255204	12616446	7757812	10493221	14035411
F(6,6)	15060832	13945687	16650459	11238256	21856192	17852248	15463182	20679496	10850578	16575939	17392324	15221558
F(6,7)	9989280	12066416	19980424	16930004	19556356	17053816	9947114	18553366	9046139	16704946	19303838	13536092
F(6,8)	11660746	11026406	16989604	15045952	19008968	18129820	14747368	16017586	16063584	15740605	10186153	12306050
F(6,9)	9046139	14267301	14080522	11978983	20473016	14468102	25235966	15987944	9655943	16321486	10683978	18536540
F(6,10)	22446342	16854500	14806036	17194002	17510924	20722958	14970886	15623101	13007723	19862446	14802673	15193104
F(6,11)	14222444	10408492	14046776	16189742	19452584	10924847	3298484.5	12731876	16582654	23677382	18951918	12307840
Fproblem	1178716160	1065109760	1260451200	1048272128	1197540608	1032963072	1204631552	1051352256	1227500288	1068153088	1228601088	1064043072
CPU(sec)	35.824176	31.428571	26.868132	22.857143	37.967033	19.450549	33.736264	27.692308	26.538462	19.230769	27.747253	52.307692

APPENDIX B

LAGRANGEAN RELAXATION, DISTAL, DISTFEAS, PROFEAS CODES

```
// Lagrangean relaxation algorithm code
// Program to perform Level1 of Hierarchical Decomposition
// Program to perform Aggregate Production-Distribution problem
// Lagrangean relaxation and subgradient procedure is used
// includes DISTAL, DISTFEAS, PROFEAS

#include <bios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloc.h>
#include <time.h>
#include <conio.h>
#include <math.h>
#include <values.h>


// Function declarations
void *alloc_vector(unsigned int elements,size_t size);
void **alloc_matrix(unsigned int rows,unsigned int columns,size_t size);
void free_matrix(float **m,unsigned int rows);
void free_matrix2(unsigned int **m,unsigned int rows);
void free_matrix3(long **m,unsigned int rows);
void read_command_file(char *pathname);
void num2str(int num,char *str);
void get_files(void);
void skip_line(FILE *file);
void open_input_files(char *pathname);
void initialize(void);
void process_iteration(unsigned int tmin);
void prepare_com01_distr(void);
void prepare_output_files(char *pathname);
void free_arrays(void);

void prepare_u(char *pathname);
void prepare_mh(char *pathname);
void initialize_u(char *pathname);

void initialize_pro_Zup(void);
void free_arrays_pro_Zup(void);
void calculate_Zup_pro(void);
void calculate_Zup_distr(void);
void compute_Zup(void);

void open_pro_input_files(char *pathname);
void initialize_pro(void);
void process_iteration_pro(void);
void prepare_pro_output_files(char *pathname);
void free_arrays_pro(void);

void process_pro(void);
void obtain_YOit(unsigned int tmin);
void initialize_YOit(void);
void present_YOit(void);

void process_inv(void);

void compute_lam(void);
```

```
void compute_u(void);
void apply_heuristic(void);
void distr(void);
void pro_inv(void);

void feasibility_check(void);
void apply_heuristic1(void);
void reorder_opik(void);
void update_all(void);
void open_read_didik(char *pathname);
void open_read_mfdh(char *pathname);
void read_output_files(char *pathname);

void initialize_u3(void);
void store_u3(void);
void copyuu3(void);

#define tmax 13
#define R 10
#define maxit 100
#define eps 0.01


// Global variables
unsigned int I;          // Number of items
unsigned int K;          // Number of customers
unsigned int T;          // Production horizon
unsigned int J;          // Number of depots

unsigned int i,k;
float **mf;              // Matrix for manuf. fix cost
float **mc;              // Matrix for manuf. variable cost
float **dh;              // Matrix for depot holding cost
float **df;              // Matrix for depot fix cost
float **dc;              // Matrix for depot variable cost
float **u;
float **u1;              // Matrix for lagrangian multipliers
float **u2;
float **u3;
long *dem;               // Array for customer demand

long **my                // Matrix for shipments from manuf.
long **di;               // Matrix for inventory levels
long **dy;               // Matrix for shipments from depots

char fin[12][13];        // Input data file names
FILE *pin[12];           // Input data file pointers
char fout[5][13];        // Output data file names
FILE *pout[5];           // Output data file pointers


float **ms;
float **mp;
float **mh;
float *ma;
float *A;

long **mx;
```

```
long **mi;


char filename[100];                  // general file name with its path
char dirname[100];
char com[13]={"com01.dat"};          // name of command file
char logfile[13]={"com03.log"};      // name of log file

FILE *cfile;                         // Command file pointer
FILE *flog;                          // Log file pointer

float Ap,Ad,Adik,Zup;

unsigned int **o;
long **YOit,**demit,**dit;
float **w;

float Z,Zup,Zlow,ZD,ZDp,Zinv,Zpro,Zdistr,Zh,Ztemp;
float lam;
unsigned int r,itr,epsilon;
float *excess;

float **wik;
unsigned int **opik, **dik;
unsigned int tp,tover;
float cumYO,cumA;


void main(int argc,char *argv[])
{
    clock_t start,end;
    start=clock();
    // Clear the screen
    clrscr();

    // Command line check
    if(argc!=2)
    {
        puts("argument count mismatch");
        exit(1);
    }
    // Open log file
    strcpy(filename,argv[1]);
    strcat(filename,logfile);
    if((flog=fopen(filename,"wt"))==NULL)
    {
        puts("Unable to open log file.");
        exit(1);
    }

    // Read command file
    read_command_file(argv[1]);
    strcpy(dirname,argv[1]);
    initialize_u(argv[1]);
    initialize_u3();
    prepare_mh(argv[1]);
    compute_Zup();
    Zlow=0.0;
```

```
epsilon=1;
ZDp=0.0;
r=0;
lam=2.0;
itr=1;
while ((itr<=maxit)&&(epsilon))
{
    printf("\niteration=%u",itr);
    initialize_YOit();
    fprintf(flog,"\n*************************************************");
    fprintf(flog,"\nITERATION=%u ",itr);
    Zdistr=0.0;
    Zpro=0.0;
    Zinv=0.0;
    Zh=0.0;
    cumYO=0.0;
    cumA=0.0;
    prepare_u(argv[1]);
    distr();
    pro_inv();
    ZD=Zdistr+Zpro+Zinv;
    fprintf(flog,"\nZdistr=%-8.2f Zpro=%-8.2f Zinv=%-8.2f Zh=%-8.2f",Zdistr,Zpro,Zinv,Zh);
    fprintf(flog,"\nZDp=%-10.2f ZD=%-10.2f Zlow=%-10.2f Zup=%-10.2f",ZDp,ZD,Zlow,Zup);
    compute_lam();
    compute_u();
    if (ZD>Zlow)
    {
        Zlow=ZD;
        feasibility_check();
        apply_heuristic();
    if (Zh<Zup)
    {
        Zup=Zh;
        store_u3();
    }
    if ((fabs(Zup-Zlow)/Zup)<=eps)
                epsilon=0;
    }
    prepare_pro_output_files(dirname);
    free_arrays_pro();
    itr++;
    ZDp=ZD;
}
end=clock();
fprintf(flog,"\n\nZlow=%f  Zup=%f",Zlow,Zup);
fprintf(flog,"\nerror=%f",(fabs(Zup-Zlow)/Zup));
fprintf(flog,"\nCPU TIME = %f ",(end-start)/CLK_TCK);
copyuu3();
initialize_YOit();
Zdistr=0.0;
Zpro=0.0;
Zinv=0.0;
Zh=0.0;
cumYO=0.0;
cumA=0.0;
distr();
pro_inv();
ZD=Zdistr+Zpro+Zinv;
```

```
        feasibility_check();
        apply_heuristic();
        prepare_pro_output_files(dirname);
        free_arrays_pro();
        free_matrix(u3,l+1);
        free_matrix(mh,l+1);
        fclose(flog);
}

// Function to allocate memory for vector
void *alloc_vector(unsigned int elements,size_t size)
{
      void *temp;
      if((temp=malloc(elements*size))==NULL)
      {
            fputs("Unable to allocate memory.",flog);
            exit(1);
      }
      return(temp);
}

// Function to allocate memory for matrix
void **alloc_matrix(unsigned int rows,unsigned int columns,size_t size)
{
      void **a;
      unsigned int i;
      if((a=malloc(rows*sizeof(void *)))==NULL)
      {
            fputs("Memory Allocation Error!",flog);
            exit(1);
      }
      for(i=0;i<rows;i++)
      {
            if((*(a+i)=malloc(columns*size))==NULL)
            {
                  fputs("Memory Allocation Error!",flog);
                  exit(1);
            }
      }
      return(a);
}

// Free previously allocated matrix memory for float values
void free_matrix(float **m,unsigned int rows)
{
      unsigned int i;
      for(i=0;i<rows;i++)free(*(m+i));
      free(m);
}

// Free previously allocated matrix memory for integer values
void free_matrix2(unsigned int **m,unsigned int rows)
{
      unsigned int i;
      for(i=0;i<rows;i++)free(*(m+i));
      free(m);
}
```

```
void free_matrix3(long **m,unsigned int rows)
{
    unsigned int i;
    for(i=0;i<rows;i++)free(*(m+i));
    free(m);
}


// Function to read characters until new line from file
void skip_line(FILE *file)
{
    char ch;
    do   ch=fgetc(file); while(ch!=10);
}

// Function to open and process command file
void read_command_file(char *pathname)
{
    // Status Message
    fputs("Processing command file...\n\n",flog);

    strcpy(filename,pathname);
    fprintf(flog,"Network directory: %s\n",filename);
    strcat(filename,com);

    // Open command file
    if((cfile=fopen(filename,"rt"))==NULL)
    {
        fputs("Unable to open command file.",flog);
        exit(1);
    }

    // Read contents of command file
    skip_line(cfile);
    fscanf(cfile,"%u",&I);
    fprintf(flog,"Number of items : %u\n",I);
    fgetc(cfile);

    skip_line(cfile);
    fscanf(cfile,"%u",&K);
    fprintf(flog,"Number of customers : %u\n",K);
    fgetc(cfile);

    skip_line(cfile);
    fscanf(cfile,"%u",&T);
    fprintf(flog,"Number of periods : %u\n",T);
    fgetc(cfile);

    skip_line(cfile);
    fscanf(cfile,"%u",&J);
    fprintf(flog,"Number of depots: %u\n",J);
    fgetc(cfile);

    fclose(cfile);
}


void num2str(int num,char *str)
```

```
{
    int hundt=0,ten=0,one=0;

    while (num>=100)
    {
        num=num-100;
        hundt++;
    }
    while (num>=10)
    {
        num=num-10;
        ten++;
    }
    one=num;

    str[0]=hundt+48;   //to convert char
    str[1]=ten+48;
    str[2]=one+48;
    str[3]=NULL;
}


void get_files(void)
{
    unsigned int m;                                    // Counter variable

    char extension[13]={".dat"};
    char total[13];
    char level[4];

    strcpy(fin[0],"mf");
    strcpy(fin[1],"mc");
    strcpy(fin[2],"dh");
    strcpy(fin[3],"df");
    strcpy(fin[4],"dc");
    strcpy(fin[5],"d");

    num2str(i,level);
    strcpy(total,level);
    num2str(k,level);
    strcat(total,level);
    for(m=0;m<6;m++)
    {
        strcat(fin[m],total);
        strcat(fin[m],extension);
    }

    strcpy(fout[0],"my");
    strcpy(fout[1],"di");
    strcpy(fout[2],"dy");

    for(m=0;m<3;m++)
    {
        strcat(fout[m],total);
        strcat(fout[m],extension);
    }
}
```

```
// Function to open input files
void open_input_files(char *pathname)
{
    unsigned int m;                                          // Counter variable

    for(m=0;m<6;m++)
    {
        strcpy(filename,pathname);
        strcat(filename,fin[m]);
        // Open file
        if((pin[m]=fopen(filename,"rt"))==NULL)
        {
            fprintf(flog,"Unable to open input file= %s",filename);
            exit(1);
        }
    }
}


// Function to initialize operation arrays
void initialize(void)
{
    unsigned int m,j;                                        // Counter variable


    // Allocate memory for input var. matrices
    mf=(float **)alloc_matrix(J+1,T+1,sizeof(float));
    mc=(float **)alloc_matrix(J+1,T+1,sizeof(float));
    dh=(float **)alloc_matrix(J+1,T+1,sizeof(float));
    df=(float **)alloc_matrix(J+1,T+1,sizeof(float));
    dc=(float **)alloc_matrix(J+1,T+1,sizeof(float));
//  u=(float **)alloc_matrix(I+1,T+1,sizeof(float));

    // Allocate memory for demand arrays
    dem=(long *)alloc_vector(T+1,sizeof(long));

    // Allocate memory for output var. matrices
    my=(long **)alloc_matrix(J+1,T+1,sizeof(long));
    di=(long **)alloc_matrix(J+1,T+1,sizeof(long));
    dy=(long **)alloc_matrix(J+1,T+1,sizeof(long));
    dik=(unsigned int  **)alloc_matrix(T+1,3,sizeof(int));

    // Initialize all arrays and matrices with zero values
    for(m=0;m<T+1;m++)
        dem[m]=0;

    for(m=0;m<J+1;m++)
        for(j=0;j<T+1;j++)
        {
            mf[m][j]=0.0;
            mc[m][j]=0.0;
            dh[m][j]=0.0;
            df[m][j]=0.0;
            dc[m][j]=0.0;
            my[m][j]=0;
            di[m][j]=0;
            dy[m][j]=0;
```

```
        }

    for(m=0;m<T+1;m++)
        for(j=0;j<3;j++)
            dik[m][j]=0;

    //Load input data
    for(m=1;m<=J;m++)
        for(j=1;j<=T;j++)
        {
            fscanf(pin[0],"%f",&mf[m][j]);
            fscanf(pin[1],"%f",&mc[m][j]);
            fscanf(pin[2],"%f",&dh[m][j]);
            fscanf(pin[3],"%f",&df[m][j]);
            fscanf(pin[4],"%f",&dc[m][j]);
            fscanf(pin[5],"%ld",&dem[j]);
        }
    for(m=1;m<=J;m++)
        for(j=1;j<=T;j++)
            mc[m][j]=mc[m][j]-mh[i][j];

    // Close input files
    for(m=0;m<6;m++)
        fclose(pin[m]);
}

// Function to form ww calculations
void process_iteration(unsigned int tmin)
{
    unsigned int t,l,a,b,c,e,m,M,jmin,count;
    float Fmin,fmin,ftemp;
    unsigned int **L[tmax];
    unsigned int **Y[tmax];
    float *F[tmax];
    unsigned int *N[tmax];
    unsigned int *n;
    float *f;


    //forward algorithm
    f=(float *)alloc_vector(T+1,sizeof(float));
    n=(unsigned int *)alloc_vector(T+1,sizeof(int));
    for(b=0;b<T+1;b++)
    {
        n[b]=0;
        f[b]=0.0;
    }
    for(b=tmin;b<T+1;b++)
        f[b]=MAXFLOAT;

    for(t=tmin;t<=T;t++)
    {
        N[t]=(unsigned int *)alloc_vector(t+1,sizeof(int));
        for(b=0;b<t+1;b++)
            N[t][b]=0;
        //  printf("%u",N[1][1]);getch();
        for(l=tmin;l<=t;l++)
        {
```

```
Fmin=MAXFLOAT;  e=0;
if (l!=t)
{
//      e=0;
        for(m=1;m<L[l][0][0];m++)
        {
            count=0;
            fmin=MAXFLOAT;
            ftemp=F[l][m];
            for(b=1;b<=J;b++)
            {
                if((Y[l][m][b]==0)&&(Y[l][0][b]==1))
                {
                    count=count+1;
                    F[l][m]=ftemp+(mc[b][l]+u[i][l])*dem[t]+dc[b][t]*dem[t]+df[b][t];
                    for(c=l;c<t;c++)
                        F[l][m]=F[l][m]+dh[b][c]*dem[t];
                    if(F[l][m]<fmin)
                    {
                        fmin=F[l][m];
                        jmin=b;
                    }
                }
            }
            if(count>0)
            {
                e=e+1;
                for (c=1; c<=t-l; c++)
                    L[l][e][c]=L[l][m][c];
                for (c=1; c<=J; c++)
                    Y[l][e][c]=Y[l][m][c];
        //      L[l][e]=L[l][m];
                L[l][e][t-l+1]=jmin;
        //      Y[l][e]=Y[l][m];
                F[l][e]=fmin;
                if (Fmin>fmin)
                {
                    Fmin=fmin;
                    M=e;
                }
            }
        }
        for(m=L[l][0][0]+1; m<=L[l][1][0]; m++)
        {
            count=0;
            fmin=MAXFLOAT;
            ftemp=F[l][m];
            for(b=1;b<=J;b++)
            {
                if((Y[l][m][b]==0)&&(Y[l][0][b]==1))
                {
                    count=count+1;
                    F[l][m]=ftemp+(mc[b][l]+u[i][l])*dem[t]+dc[b][t]*dem[t]+df[b][t];
                    for(c=l;c<t;c++)
                        F[l][m]=F[l][m]+dh[b][c]*dem[t];
                    if(F[l][m]<fmin)
                    {
                        fmin=F[l][m];
```

```
                        jmin=b;
                    }
                }
            }
            if(count>0)
            {
                e=e+1;
                for (c=1; c<=t-l; c++)
                    L[l][e][c]=L[l][m][c];
                for (c=1; c<=J; c++)
                    Y[l][e][c]=Y[l][m][c];
            //  L[l][e]=L[l][m];
                L[l][e][t-l+1]=jmin;
            //  Y[l][e]=Y[l][m];
                F[l][e]=fmin;
                if (Fmin>fmin)
                {
                    Fmin=fmin;
                    M=e;
                }
            }
        }
        for (m=1; m<=J; m++)
        {
            e=e+1;
            for (c=1; c<=t-l; c++)
                L[l][e][c]=L[l][0][c];
            for (c=1; c<=J; c++)
                Y[l][e][c]=Y[l][0][c];
            L[l][e][t-l+1]=m;
            F[l][e]=F[l][0]+Y[l][0][m]*mf[m][l]+(mc[m][l]+u[i][l])*dem[t]+
                                                    dc[m][t]*dem[t]+df[m][t];
            for (c=l; c<t; c++)
                F[l][e]=F[l][e]+dh[m][c]*dem[t];
            if (dem[t]>0)
                Y[l][e][m]=0;
            if (Fmin>F[l][e])
            {
                Fmin=F[l][e];
                M=e;
            }
        }
    }
    else
    {
        a=T-l+1;
        L[l]=(unsigned int **)alloc_matrix(J*a-a+2,a+1,sizeof(int));
        F[l]=(float *)alloc_vector(J*a-a+2,sizeof(float));
        Y[l]=(unsigned int **)alloc_matrix(J*a-a+2,J+1,sizeof(int));
        for (b=0; b<J*a-a+2; b++)
        {
            for (c=0; c<a+1; c++)
                L[l][b][c]=0;
            F[l][b]=0.0;
        }
        for (b=0; b<J*a-a+2; b++)
            for (c=0; c<J+1; c++)
                Y[l][b][c]=1;
```

```
                for (b=1; b<=J; b++)
                    L[l][b][1]=b;
                if (dem[t]>0)
                    for (b=1; b<=J; b++)
                        Y[l][b][b]=0;
                for (m=1; m<=J; m++)
                {
                    F[l][m]=mf[m][l]+(mc[m][l]+u[i][l])*dem[t]+dc[m][t]*dem[t]+df[m][t];
                    if (Fmin>F[l][m])
                    {
                        Fmin=F[l][m];
                        M=m;
                    }
                }
                e=J;
            }
            for (b=1; b<=t-l+1; b++)
            {
                L[l][0][b]=L[l][M][b];
            }
            for (b=1; b<=J; b++)
            {
                Y[l][0][b]=Y[l][M][b];
            }
            L[l][0][0]=M;
            L[l][1][0]=e;
            F[l][0]=F[l][M];
            if ((Fmin+f[l-1])<f[t])
            {
                f[t]=Fmin+f[l-1];
                n[t]=l;
                for (m=1; m<=l-1; m++)
                    N[t][m]=N[l-1][m];
                for (m=l; m<=t; m++)
                    N[t][m]=L[l][0][m-l+1];
            }
        }
    }
    Zdistr=Zdistr+f[T];

    b=T;
    do
    {
        c=n[b];
////    fprintf(flog,"\nt=%u l=%u",b,c);
        for (m=c; m<=b; m++)
        {
////        fprintf(flog," j=%u",N[b][m]);
            dik[m][1]=c;
            dik[m][2]=N[b][m];
            my[(N[b][m])][c]=my[(N[b][m])][c]+dem[m];
            dy[(N[b][m])][m]=dem[m];
            for (a=m-1; a>=c; a--)
                di[(N[b][m])][a]=di[(N[b][m])][a]+dem[m];
        }
        b=c-1;
    }
    while (b!=(tmin-1));
```

```
    for (b=tmin; b<=T; b++)
    {
        a=T-b+1;
        free_matrix2(L[b],J*a-a+2);
        free_matrix2(Y[b],J*a-a+2);
        free(F[b]);
    }
    free(f);

    free(n);
    for (b=tmin; b<=T; b++)
        free(N[b]);
    for (m=1;m<=J;m++)
        for (t=tmin;t<=T;t++)
        {
            if (my[m][t]>0)
                Zh+=mf[m][t]+mc[m][t]*my[m][t];
            Zh+=dh[m][t]*di[m][t];
            if (dy[m][t]>0)
                Zh+=df[m][t]+dc[m][t]*dy[m][t];
        }
}


void prepare_output_files(char *pathname)
{
    unsigned int m,t;                                   // Counter variable

    for(m=0;m<3;m++)
    {
        strcpy(filename,pathname);
        strcat(filename,fout[m]);
        // Open file
        if((pout[m]=fopen(filename,"wt"))==NULL)
        {
            fprintf(flog,"Unable to open output file=%s",filename);
            exit(1);
        }
    }
    for(m=1;m<J+1;m++)
    {
        for(t=1;t<T+1;t++)
        {
            fprintf(pout[0],"%ld ",my[m][t]);
            fprintf(pout[1],"%ld ",di[m][t]);
            fprintf(pout[2],"%ld ",dy[m][t]);
        }
        fprintf(pout[0],"\n");
        fprintf(pout[1],"\n");
        fprintf(pout[2],"\n");
    }

    fprintf(pout[1],"\n");
    for(m=1;m<T+1;m++)
    {
        for(t=1;t<3;t++)
        {
```

```
                fprintf(pout[1],"%u ",dik[m][t]);
            }
            fprintf(pout[1],"\n");
        }
        for(m=0;m<3;m++)
            fclose(pout[m]);
}


void prepare_com01_distr(void)
{
    unsigned int m,t;                                   // Counter variable

    fprintf(flog,"\nmy table\n");
    for(m=1;m<J+1;m++)
    {
        for(t=1;t<T+1;t++)
            fprintf(flog,"%-8ld ",my[m][t]);
        fprintf(flog,"\n");
    }
    fprintf(flog,"\ndi table\n");
    for(m=1;m<J+1;m++)
    {
        for(t=1;t<T+1;t++)
            fprintf(flog,"%-8ld ",di[m][t]);
        fprintf(flog,"\n");
    }
    fprintf(flog,"\ndy table\n");
    for(m=1;m<J+1;m++)
    {
        for(t=1;t<T+1;t++)
            fprintf(flog,"%-8ld ",dy[m][t]);
        fprintf(flog,"\n");
    }

}


// Function to free memory allocated for arrays and matrices
void free_arrays(void)
{
    // Free allocated memory for input arrays

    free(dem);

    // Free allocate memory for matrices
    free_matrix(mf,J+1);
    free_matrix(mc,J+1);
    free_matrix(dh,J+1);
    free_matrix(df,J+1);
    free_matrix(dc,J+1);
    free_matrix3(my,J+1);
    free_matrix3(di,J+1);
    free_matrix3(dy,J+1);
    free_matrix2(dik,T+1);
}
```

```
void open_pro_input_files(char *pathname)

{
    unsigned int m;                                          // Counter variable

//  strcpy(fin[6],"u.dat");
    strcpy(fin[7],"ms.dat");
    strcpy(fin[8],"mp.dat");
    strcpy(fin[9],"mh.dat");
    strcpy(fin[10],"ma.dat");
    strcpy(fin[11],"A.dat");

    strcpy(fout[3],"mx.dat");
    strcpy(fout[4],"mi.dat");

    for(m=7;m<12;m++)
    {
        strcpy(filename,pathname);
        strcat(filename,fin[m]);
        // Open file
        if((pin[m]=fopen(filename,"rt"))==NULL)
        {
            fprintf(flog,"Unable to open input file= %s",filename);
            exit(1);
        }
    }
}


void initialize_pro(void)
// Function to initialize operation arrays
{
    unsigned int m,j;                                        // Counter variable

    // Allocate memory for input var. matrices
    ms=(float **)alloc_matrix(I+1,T+1,sizeof(float));
    mp=(float **)alloc_matrix(I+1,T+1,sizeof(float));

    ma=(float *)alloc_vector(I+1,sizeof(float));

    // Allocate memory for demand arrays
    A=(float *)alloc_vector(T+1,sizeof(float));

    // Allocate memory for output var. matrices
    mx=(long **)alloc_matrix(I+1,T+1,sizeof(long));
    mi=(long **)alloc_matrix(I+1,T+1,sizeof(long));

    w=(float **)alloc_matrix(I+1,T+1,sizeof(float));
    o=(unsigned int **)alloc_matrix(I+1,T+1,sizeof(unsigned int));
    dit=(long**)alloc_matrix(I+1,T+1,sizeof(long));

    excess=(float *)alloc_vector(T+1,sizeof(float));


    // Initialize all arrays and matrices with zero values
    for(m=0;m<I+1;m++)
    {
```

```
            for(j=0;j<T+1;j++)
            {
                ms[m][j]=0.0;
                mp[m][j]=0.0;
                A[j]=0.0;
            }
            ma[m]=0.0;
        }

        for(m=0;m<I+1;m++)
            for(j=0;j<T+1;j++)
            {
                mx[m][j]=0;
                mi[m][j]=0;
            }

        for(m=0;m<I+1;m++)
            for(j=0;j<T+1;j++)
            {
                w[m][j]=0.0;
                o[m][j]=m;
                dit[m][j]=0;
            }

        for (j=0;j<T+1;j++)
            excess[j]=0.0;

        //Load input data
        for(m=1;m<=I;m++)
        {
            for(j=1;j<=T;j++)
            {
                fscanf(pin[7],"%f",&ms[m][j]);
                fscanf(pin[8],"%f",&mp[m][j]);
                fscanf(pin[11],"%f",&A[j]);
            }
            fscanf(pin[10],"%f",&ma[m]);
        }
        for(m=1;m<=I;m++)
            for(j=1;j<=T;j++)
                mp[m][j]=mp[m][j]+mh[m][j];

        // Close input files
        for(m=7;m<12;m++)
            fclose(pin[m]);
}


void prepare_pro_output_files(char *pathname)
{
        unsigned int m,t;                                   // Counter variable

        for(m=3;m<5;m++)
        {
            strcpy(filename,pathname);
            strcat(filename,fout[m]);
            // Open file
            if((pout[m]=fopen(filename,"wt"))==NULL)
```

```
            {
                fprintf(flog,"Unable to open pro-output file=%s",filename);
                exit(1);
            }
        }

        strcpy(filename,pathname);
        strcat(filename,fin[6]);
            // Open file
        if((pin[6]=fopen(filename,"wt"))==NULL)
        {
            fputs("Unable to open output file.",flog);
            exit(1);
        }

        for(m=1;m<I+1;m++)
        {
            for(t=1;t<T+1;t++)
            {
                fprintf(pout[3],"%ld ",mx[m][t]);
                fprintf(pout[4],"%ld ",mi[m][t]);
                fprintf(pin[6],"%-10.2f",u1[m][t]);
            }
            fprintf(pout[3],"\n");
            fprintf(pout[4],"\n");
            fprintf(pin[6],"\n");
        }
        free_matrix(u1,I+1);
        for(m=3;m<5;m++)
            fclose(pout[m]);
        fclose(pin[6]);
}


void free_arrays_pro(void)
{
        // Free allocated memory for input arrays
        free(A);
        free(ma);

        // Free allocate memory for matrices
        free_matrix(ms,I+1);
        free_matrix(mp,I+1);
        free_matrix(u,I+1);
        free_matrix(u2,I+1);
        free_matrix(w,I+1);
        free_matrix2(o,I+1);
        free_matrix3(YOit,I+1);
        free_matrix3(demit,I+1);
        free_matrix3(dit,I+1);
        free_matrix3(mx,I+1);
        free_matrix3(mi,I+1);
        free(excess);
}


void initialize_YOit(void)
{
```

```
    unsigned int m,n;

    YOit=(long **)alloc_matrix(I+1,T+1,sizeof(long));
    demit=(long **)alloc_matrix(I+1,T+1,sizeof(long));
    for (m=0;m<I+1;m++)
        for (n=0;n<T+1;n++)
        {
            YOit[m][n]=0;
            demit[m][n]=0;
        }
}


void obtain_YOit(unsigned int tmin)
{
    unsigned int m,n;

    for (m=tmin;m<=T;m++)
        for (n=1;n<=J;n++)
            YOit[i][m]=YOit[i][m]+my[n][m];

    for (m=tmin;m<=T;m++)
        demit[i][m]=demit[i][m]+dem[m];
}


void process_pro(void)
{
    unsigned int t,j,imin;
    float minw;

    for (t=1;t<=T;t++)
    {
        minw=MAXFLOAT;
        for (j=1;j<=I;j++)
        {
            w[j][t]=(mp[j][t]-u[j][t])/ma[j]+ms[j][t]/A[t];
            if (w[j][t]<minw)
            {
                minw=w[j][t];
                imin=j;
            }
        }
        if (minw<0)
        {
            mx[imin][t]=A[t]/ma[imin];
            Zpro+=mx[imin][t]*(mp[imin][t]-u[imin][t])+ms[imin][t];
        }
        else
            excess[t]=A[t];
    }
    fprintf(flog,"mx Table\n");
    for (j=1;j<=I;j++)
    {
        for (t=1;t<=T;t++)
            fprintf(flog,"%-8ld",mx[j][t]);
        fprintf(flog,"\n");
    }
```

```
}

void compute_lam(void)
{
    if (ZD>ZDp)
    {
        if (itr%10==0)
            lam=lam/2;
    }
    else
    {
        r=r+1;
        if (r==R)
        {
            r=0;
            lam=lam/2;
        }
    }
    fprintf(flog,"\nlambda=%f",lam);
}

void compute_u(void)
{
    unsigned int j,t;
    float totsub;
    long **sg;
    float s;    //steplength

    totsub=0.0;
    sg=(long **)alloc_matrix(I+1,T+1,sizeof(long));
    u1=(float **)alloc_matrix(I+1,T+1,sizeof(float));

    for (j=0;j<I+1;j++)
    {
        for (t=0;t<T+1;t++)
        {
            sg[j][t]=0;
            u1[j][t]=0.0;
        }
    }
    fprintf(flog,"\nsubgradient table\n");
    for (j=1;j<=I;j++)
    {
     for (t=1;t<=T;t++)
     {
            sg[j][t]=sg[j][t-1]+YOit[j][t]-mx[j][t];
     //     printf("\nsg[%u][%u]=%ld    ",j,t,sg[j][t]);
            totsub=totsub+1.00*sg[j][t]*sg[j][t];
     //     printf("\ntotsub=%f",totsub); getch();
            fprintf(flog,"%-8ld",sg[j][t]);
        }
        fprintf(flog,"\n");
    }
    fprintf(flog,"total subgradient=%f",totsub);
```

```
//    printf("\nabs=%f    ",(Zup-ZD)); getch();
      s=lam*(Zup-ZD)/totsub;
      fprintf(flog,"\nsteplength=%f",s);
      fprintf(flog,"\nu[i][t] TABLE\n");
      for(j=1;j<=I;j++)
      {
            for(t=1;t<=T;t++)
            {
                  u1[j][t]=u2[j][t]+s*sg[j][t];
                  if (u1[j][t]<0)
                        u1[j][t]=0.0;
                  fprintf(flog,"%-10.2f",u1[j][t]);
            }
            fprintf(flog,"\n");
      }
      free_matrix3(sg,I+1);
}


void apply_heuristic(void)
{
      unsigned int t,j,b,c,e,m,n,l,imin,d;
      long inv,norm,q,qd, *def;
      float *vim,maxvim;
      unsigned int *dl;
      unsigned int *item;
      float *load;
      float temp1,vimtemp;

      dl=(unsigned int *)alloc_vector(I+1,sizeof(int));
      def=(long *)alloc_vector(I+1,sizeof(long));
      vim=(float *)alloc_vector(I+1,sizeof(float));
      item=(unsigned int *)alloc_vector(T+1,sizeof(int));
      load=(float *)alloc_vector(T+1,sizeof(float));

      for (j=0;j<I+1;j++)
      {
            dl[j]=0;
            def[j]=0;
            vim[j]=0.0;
            for (t=0;t<T+1;t++)
                  mi[j][t]=0;
      }
      for (t=0;t<T+1;t++)
      {
            item[t]=0;
            load[t]=0.0;
      }
      fprintf(flog,"\n\nLAGRANGEAN HEURISTIC OF PRO. PROB.");
      Z=Zdistr+Zpro;
      Ztemp=Z;
      for (t=1;t<=T;t++)
      {
            fprintf(flog,"\nt=%u  ",t);
            e=0;
            for (j=0;j<I+1;j++)
                  vim[j]=0.0;
            for (b=1;b<=I;b++)
```

```
{
    inv=mi[b][t-1]+mx[b][t]-YOit[b][t];
    fprintf(flog,"inv[%u]=%-9ld",b,inv);
    if (inv<0)
    {
        e=e+1;
        dl[e]=b;
        def[e]=-inv;
    }
    else
    {
        mi[b][t]=inv;
    }
}
dl[0]=e;
for (j=1;j<=I;j++)
{
    if (mi[j][t-1]>0)
    {
        if (mi[j][t-1]>YOit[j][t])
            temp1=YOit[j][t]*ma[j];
        else
            temp1=mi[j][t-1]*ma[j];
        l=1;
        while (temp1>0.0)
        {
            while (item[l]!=j)
                l++;
                if (load[l]>temp1)
                {
                    load[l]-=temp1;
                    temp1=0.0;
                }
                else
                {
                    temp1-=load[l];
                    load[l]=0.0;
                    item[l]=0;
                    l++;
                }
        }
    }
    if (mx[j][t]>0)
    {
        if (mi[j][t-1]>YOit[j][t])
        {
            load[t]=mx[j][t]*ma[j];
            item[t]=j;
        }
        else
        {
            if (mi[j][t]>0)
            {
                load[t]=mi[j][t]*ma[j];
                item[t]=j;
            }
        }
    }
```

```
        }
        while (dl[0]>0)
        {
            maxvim=MAXFLOAT;
            for (m=1;m<=dl[0];m++)
            {
                vim[m]=0.0;
                n=dl[m];
//              fprintf(flog,"\ni=%u",n);
                l=t;
                norm=def[m];
                while (norm>0)
                {
//                  fprintf(flog," excess[%u]=%-8.2f",l,excess[l]);
//                  fprintf(flog,"load[%u]=%-8.2f",l,load[l]);
                    if (excess[l]>=ma[n])
                    {
                        q=excess[l]/ma[n];
                        if (q>norm)
                            q=norm;
                //      fprintf(flog," q=%ld ",q);
                        if (mx[n][l]==0)
                            vim[m]=vim[m]+ms[n][l];
                        vimtemp=(mp[n][l]-u[n][l])*q;
                        vim[m]+=vimtemp;
                        norm=norm-q;
                    }

                    if ((load[l]>=ma[n])&&(norm>0))
                    {
                        q=load[l]/ma[n];
                        if (q>norm)
                                q=norm;
                        d=item[l];
                        qd=ceil(q*ma[n]/ma[d]);
                    //  fprintf(flog," q=%ld ",q);
                        if (mx[n][l]==0)
                            vim[m]=vim[m]+ms[n][l];
                        vimtemp=(mp[n][l]-u[n][l])*q;
                        vim[m]+=vimtemp;
                        if (qd==mx[d][l])
                            vim[m]-=ms[d][l];
                        vimtemp=(mp[d][l]-u[d][l])*qd;
                        vim[m]-=vimtemp;
                        norm=norm-q;
                    }

                    l--;
                }
                vimtemp=def[m]*ma[n];
                vim[m]=vim[m]/vimtemp;
//              fprintf(flog,"i=%u vim[%u]=%-8.2f",n,m,vim[m]);
                if (vim[m]<maxvim)
                {
                    maxvim=vim[m];
                    imin=m;
                }
            }
```

```
                n=dl[imin];
                l=t;
//              fprintf(flog,"\nminvim=%-8.2f,i=%u",maxvim,n);
//              fprintf(flog,"  Added cost(Cost inc-Cost dec)=%-10.2f
",(vim[imin]*(def[imin]*ma[dl[imin]])));
                vimtemp=def[imin]*ma[n];
                vimtemp=vimtemp*vim[imin];
                Z+=vimtemp;

                while (def[imin]>0)
                {
                    if (excess[l]>=ma[n])
                    {
                        q=excess[l]/ma[n];
                        if (q>def[imin])
                            q=def[imin];
                        mx[n][l]=mx[n][l]+q;
                        for (c=l;c<=t-1;c++)
                            mi[n][c]=mi[n][c]+q;
                        temp1=q*ma[n];
                        excess[l]-=temp1;
                        def[imin]=def[imin]-q;
                    }
                        if ((load[l]>=ma[n])&&(def[imin]>0))
                        {
                            q=load[l]/ma[n];
                            if (q>def[imin])
                                q=def[imin];
                            d=item[l]; //fprintf(flog,"\nl=%u overloaded item=%u",l,item[l]);
                            qd=ceil(q*ma[n]/ma[d]);
                            mx[n][l]=mx[n][l]+q;
                            //fprintf(flog," mx[%u][%u]=%ld",d,l,mx[d][l]);
                            mx[d][l]-=qd;
                            //fprintf(flog," mx[%u][%u]=%ld",d,l,mx[d][l]);
                            //fprintf(flog,"\nmi[%u][%u]=%ld",d,l,mi[d][l]);
                            for (c=l;c<=t-1;c++)
                            {
                                mi[n][c]=mi[n][c]+q;
                                mi[d][c]-=qd;
                        //      fprintf(flog," mi[%u][%u]=%ld",d,c,mi[d][c]);
                            }
                            mi[d][t]-=qd;
                        //  fprintf(flog," mi[%u][%u]=%ld",d,t,mi[d][t]);
                            temp1=qd*ma[d];
                            load[l]-=temp1;
                            temp1=qd*ma[d];
                            excess[l]+=temp1;
                            temp1=q*ma[n];
                            excess[l]-=temp1;
                            def[imin]=def[imin]-q;
                            if (load[l]==0)
                                item[l]=0;
                        }

//              fprintf(flog,"\nexcess[%u]=%-8.2f, load[%u]=%-8.2f",l,excess[l],l,load[l]);
                    l--;
                }
                dl[imin]=0;
```

```
            for (m=imin; m<=dl[0]-1; m++)
            {
                dl[m]=dl[m+1];
                def[m]=def[m+1];
            }
            dl[0]=dl[0]-1;
        }
    }
    fprintf(flog,"\nZ(before pro. heuristic)=%-10.2f Z(after pro. heuristic)=%-10.2f",Ztemp,Z);

    for (t=1;t<=T;t++)
    {
        j=item[t]; //fprintf(flog,"\nitem=%u",j);
        if (j!=0)
        {
            q=load[t]/ma[j];  //fprintf(flog," q=%ld",q);
            load[t]-=q*ma[j];
            temp1=q*(mp[j][t]-u[j][t]);
            Z-=temp1;
            mx[j][t]-=q;
            if (mx[j][t]==0)
                Z-=ms[j][t];
            //fprintf(flog," mi[%u][%u]=%ld",j,t,mi[j][t]);
            for (l=t; l<=T; l++)
            {
                mi[j][l]-=q;
                //fprintf(flog," mi[%u][%u]=%ld",j,l,mi[j][l]);
            }
            if (load[t]==0.0)
                j=0;
        }
    }
    fprintf(flog,"\nmx Table\n");
    for (j=1;j<=I;j++)
    {
        for (t=1;t<=T;t++)
            fprintf(flog,"%-8ld",mx[j][t]);
        fprintf(flog,"\n");
    }

    fprintf(flog,"mi Table\n");
    for (m=1;m<=I;m++)
    {
        for (t=1;t<=T;t++)
            fprintf(flog,"%-8ld",mi[m][t]);
        fprintf(flog,"\n");
    }
    free(dl);
    free(def);
    free(vim);
    free(item);
    free(load);

    for (m=1;m<=I;m++)
        for (t=1;t<=T;t++)
            if (mx[m][t]>0)
                Zh+=ms[m][t]+mp[m][t]*mx[m][t];
    fprintf(flog,"\nZh=%-10.2f",Zh);
```

```
}

void distr(void)
{
    for(i=1; i<=I; i++)
        for(k=1; k<=K; k++)
        {
            get_files();
            open_input_files(dirname); // Open the 7 input files
            initialize();          // Allocate and load variable arrays
            process_iteration(1);
            obtain_YOit(1);
            prepare_output_files(dirname);
//          prepare_com01_distr();
            free_arrays();
        }
    present_YOit();
}

void pro_inv(void)
{
    open_pro_input_files(dirname);
    initialize_pro();
    process_pro();
}

void present_YOit(void)
{
    unsigned int t,j;

    fprintf(flog,"\nYOit Table\n");
    for (j=1;j<=I;j++)
    {
        for (t=1;t<=T;t++)
            fprintf(flog,"%-8ld",YOit[j][t]);
        fprintf(flog,"\n");
    }

}

void prepare_mh(char *pathname)
{
    unsigned int m,j;

    strcpy(fin[9],"mh.dat");
    strcpy(filename,pathname);
    strcat(filename,fin[9]);
    if ((pin[9]=fopen(filename,"rt"))==NULL)
    {
        fprintf(flog,"Unable to open input file=%s",filename);
        exit(1);
    }
    mh=(float **)alloc_matrix(I+1,T+1,sizeof(float));
    for (m=0;m<I+1;m++)
```

```
            for (j=0;j<T+1;j++)
                mh[m][j]=0.0;
        for (m=1;m<=I;m++)
            for (j=1;j<=T;j++)
                fscanf(pin[9],"%f",&mh[m][j]);
        for (m=1;m<=I;m++)
            for (j=T-1;j>=1;j--)
            {
                mh[m][j]+=mh[m][j+1];
//              printf("\nmh[%u][%u]=%f",m,j,mh[m][j]); getch();
            }
        fclose(pin[9]);
}

void prepare_u(char *pathname)
{
        unsigned int m,j;

        strcpy(fin[6],"u.dat");
        strcpy(filename,pathname);
        strcat(filename,fin[6]);
        if ((pin[6]=fopen(filename,"rt"))==NULL)
        {
            fprintf(flog,"Unable to open input file=%s",filename);
            exit(1);
        }
        u2=(float **)alloc_matrix(I+1,T+1,sizeof(float));
        u=(float **)alloc_matrix(I+1,T+1,sizeof(float));
        for (m=0;m<I+1;m++)
            for (j=0;j<T+1;j++)
            {
                u[m][j]=0.0;
                u2[m][j]=0.0;
            }
        for (m=1;m<=I;m++)
            for (j=1;j<=T;j++)
                fscanf(pin[6],"%f",&u2[m][j]);
        for (m=1;m<=I;m++)
            u[m][T]=u2[m][T];
        for (m=1;m<=I;m++)
            for (j=T-1;j>=1;j--)
                u[m][j]=u2[m][j]+u[m][j+1];
        fclose(pin[6]);
}

void feasibility_check(void)
{
        unsigned int n;
        float temp1;

        Ztemp=Zdistr+Zpro;
        fprintf(flog,"\nFEASIBILITY CHECK");
        for (tp=1;tp<=T;tp++)
        {
            for (n=1;n<=I;n++)
            {
                temp1=YOit[n][tp]*ma[n];
                cumYO+=temp1;
```

```
            }
            cumA+=A[tp];
            fprintf(flog,"\nt=%u cum. req.=%f  cum. cap.=%f",tp,cumYO,cumA);
            if (cumYO>cumA)
                apply_heuristic1();
        }
        present_YOit();
        Z=Zdistr+Zpro;
        fprintf(flog,"\nZ(before distr. heuristic)=%-10.2f Z(after distr. heuristic)=%-10.2f",Ztemp,Z);
}

void apply_heuristic1(void)
{
        unsigned int m,j,count;
        long q;

        opik=(unsigned int **)alloc_matrix(I*K+1,4,sizeof(int));
        wik=(float **)alloc_matrix(I+1,K+1,sizeof(float));
        for (m=0;m<I*K+1;m++)
            for (j=0;j<4;j++)
                opik[m][j]=0;
        for (m=0;m<I+1;m++)
            for (j=0;j<K+1;j++)
                wik[m][j]=0.0;
        fprintf(flog,"\n\nLAGRANGEAN HEURISTIC OF DISTR. PROB.");
        fprintf(flog,"\nWeights related to product-customer pair:\n");
        for (i=1;i<=I;i++)
            for (k=1;k<=K;k++)
            {
                open_read_didik(dirname);
                tover=dik[tp+1][1];
                free_matrix3(di,J+1);
                free_matrix2(dik,T+1);
                if (tover<=tp)
                {
                    get_files();
                    open_input_files(dirname);
                    initialize();
                    free_matrix3(di,J+1);
                    free_matrix2(dik,T+1);
                    open_read_didik(dirname);
                    reorder_opik();
                    free_arrays();
                    opik[0][0]++;
                }
            }
        fprintf(flog,"\n");
        m=0;
        while (cumYO>cumA)
        {
            m++;
            i=opik[m][2];
            k=opik[m][3];
            q=opik[m][0];
            tover=opik[m][1];
            fprintf(flog,"i=%u k=%u q=%ld tover=%u,  ",i,k,q,tover);
            get_files();
            open_input_files(dirname);
```

```
            initialize();
            read_output_files(dirname);
            update_all();
            process_iteration(tp+1);
            obtain_YOit(tp+1);
            prepare_output_files(dirname);
            free_arrays();
            cumYO-=q*ma[i];
            YOit[i][tover]-=q;
//          present_YOit();
        }
//      present_YOit();
        free_matrix2(opik,I*K+1);
        free_matrix(wik,I+1);
}

void reorder_opik(void)
{
        unsigned int *jik,t,m,found,b,ii,kk;
        long q;
        float wiktemp;

        jik=(unsigned int *)alloc_vector(J+1,sizeof(int));
        for (m=0;m<J+1;m++)
                jik[m]=1;
        for (m=tover;m<=tp;m++)
                jik[(dik[m][2])]=0;
        q=0;
        m=tp+1;
        do
        {
                q+=dem[m];
                b=dik[m][2];
//              fprintf(flog,"\nm=%u dem=%ld j=%u",m,dem[m],b);
                wik[i][k]+=jik[b]*mf[b][tover];
                wiktemp=dem[m]*mc[b][tover];
                wik[i][k]+=wiktemp+df[b][m];
                wiktemp=dem[m]*dc[b][m];
                wik[i][k]+=wiktemp;
                for (t=tover; t<=tp; t++)
                {
                        wiktemp=dem[m]*dh[b][t];
                        wik[i][k]+=wiktemp;
                }
                jik[b]=0;
                m++;
        }
        while ((dik[m][1]<=tp)&&(m<=T));
        wiktemp=q*ma[i];
        wik[i][k]=wik[i][k]/wiktemp;
        fprintf(flog,"wik(%u,%u)=%f ",i,k,wik[i][k]);
        free(jik);
        found=0;
        m=1;
        while ((found==0)&&(m<=opik[0][0]))
        {
                ii=opik[m][2];
                kk=opik[m][3];
```

```
            if (wik[i][k]>wik[ii][kk])
                found=1;
            else
                m++;
        }
        for (b=opik[0][0];b>=m;b--)
        {
            opik[b+1][0]=opik[b][0];
            opik[b+1][1]=opik[b][1];
            opik[b+1][2]=opik[b][2];
            opik[b+1][3]=opik[b][3];
        }
        opik[m][0]=q;
        opik[m][1]=tover;
        opik[m][2]=i;
        opik[m][3]=k;
}

void update_all(void)
{
    unsigned int j,t;
    float temp1;

    for (j=1;j<=J;j++)
        if (di[j][tp]>0)
        {
            for (t=tover;t<=tp;t++)
            {
                temp1=dh[j][t]*di[j][tp];
                Zh-=temp1;
                Zdistr-=temp1;
            }
            if (di[j][tp]==my[j][tover])
            {
                Zh-=mf[j][tover];
                Zdistr-=mf[j][tover];
            }
            temp1=di[j][tp]*mc[j][tover];
            Zh-=temp1;
            temp1=di[j][tp]*(mc[j][tover]+u[i][tover]);
            Zdistr-=temp1;
        }
    for (j=1;j<=J;j++)
    {
        for (t=tover;t<=tp;t++)
            di[j][t]-=di[j][tp];
        my[j][tover]-=di[j][tp];
    }
    for (t=tp+1;t<=T;t++)
    {
        for (j=1;j<=J;j++)
        {
            if (dy[j][t]>0)
            {
                Zh-=df[j][t];
                temp1=dy[j][t]*dc[j][t];
                Zh-=temp1;
                Zdistr-=df[j][t];
```

```
                    Zdistr-=temp1;
                }
                temp1=di[j][t]*dh[j][t];
                Zh-=temp1;
                Zdistr-=temp1;
                if (my[j][t]>0)
                {
                    Zh-=mf[j][t];
                    temp1=my[j][t]*mc[j][t];
                    Zh-=temp1;
                    Zdistr-=mf[j][t];
                    temp1=my[j][t]*(mc[j][t]+u[i][t]);
                    Zdistr-=temp1;
                }
                YOit[i][t]-=my[j][t];
                my[j][t]=0;
                di[j][t]=0;
                dy[j][t]=0;
            }
            dik[t][1]=0;
            dik[t][2]=0;
        }
}

void open_read_didik(char *pathname)
{
    unsigned int m,j;
    char extension[13]={".dat"};
    char total[13];
    char level[4];

    strcpy(fout[1],"di");
    num2str(i,level);
    strcpy(total,level);
    num2str(k,level);
    strcat(total,level);
    strcat(fout[1],total);
    strcat(fout[1],extension);
    strcpy(filename,pathname);
    strcat(filename,fout[1]);
    if ((pout[1]=fopen(filename,"rt"))==NULL)
    {
        fprintf(flog,"Unable to open output file=%s",filename);
        exit(1);
    }
    di=(long **)alloc_matrix(J+1,T+1,sizeof(long));
    dik=(unsigned int **)alloc_matrix(T+1,3,sizeof(int));
    for (m=0;m<J+1;m++)
        for (j=0;j<T+1;j++)
            di[m][j]=0;
    for (m=0;m<T+1;m++)
        for (j=0;j<3;j++)
            dik[m][j]=0;
    for (m=1;m<=J;m++)
        for (j=1;j<=T;j++)
            fscanf(pout[1],"%ld",&di[m][j]);
    for (m=1;m<=T;m++)
        for (j=1;j<=2;j++)
```

```
                fscanf(pout[1],"%u",&dik[m][j]);
        fclose(pout[1]);
}

void open_read_mfdh(char *pathname)
{
      unsigned int m,j;
      char extension[13]={".dat"};
      char total[13];
      char level[4];

      strcpy(fin[0],"mf");
      strcpy(fin[2],"dh");
      num2str(i,level);
      strcpy(total,level);
      num2str(k,level);
      strcat(total,level);
      strcat(fin[0],total);
      strcat(fin[0],extension);
      strcpy(filename,pathname);
      strcat(filename,fin[0]);
      if ((pin[0]=fopen(filename,"rt"))==NULL)
      {
            fprintf(flog,"Unable to open input file=%s",filename);
            exit(1);
      }
      strcat(fin[2],total);
      strcat(fin[2],extension);
      strcpy(filename,pathname);
      strcat(filename,fin[2]);
      if ((pin[2]=fopen(filename,"rt"))==NULL)
      {
            fprintf(flog,"Unable to open input file=%s",filename);
            exit(1);
      }
      mf=(float **)alloc_matrix(J+1,T+1,sizeof(float));
      dh=(float **)alloc_matrix(J+1,T+1,sizeof(float));
      for (m=0;m<J+1;m++)
            for (j=0;j<T+1;j++)
            {
                  mf[m][j]=0.0;
                  dh[m][j]=0.0;
            }
      for (m=1;m<=J;m++)
            for (j=1;j<=T;j++)
            {
                  fscanf(pin[0],"%f",&mf[m][j]);
                  fscanf(pin[2],"%f",&dh[m][j]);
                  printf("dh=%f ",dh[m][j]);

            }
      fclose(pin[0]);
      fclose(pin[2]);
}

void read_output_files(char *pathname)
{
      unsigned int m,t;
```

```
        for (m=0;m<3;m++)
        {
            strcpy(filename,pathname);
            strcat(filename,fout[m]);
            if ((pout[m]=fopen(filename,"rt"))==NULL)
            {
                fprintf(flog,"Unable to open output file=%s",filename);
                exit(1);
            }
        }
        for(m=1;m<J+1;m++)
            for(t=1;t<T+1;t++)
            {
                fscanf(pout[0],"%ld",&my[m][t]);
                fscanf(pout[1],"%ld",&di[m][t]);
                fscanf(pout[2],"%ld",&dy[m][t]);
            }
        for(m=1;m<T+1;m++)
            for(t=1;t<3;t++)
                fscanf(pout[1],"%u",&dik[m][t]);
        for(m=0;m<3;m++)
            fclose(pout[m]);
}


void compute_Zup(void)
{
        fprintf(flog,"\nZup CALCULATION");
        initialize_YOit();
        Zdistr=0.0;
        Zpro=0.0;
        Zh=0.0;
        prepare_u(dirname);
        distr();
        pro_inv();
        feasibility_check();
        apply_heuristic();
        Zup=Zh;
        fprintf(flog,"\nZup=%f",Zup);
        free_arrays_pro();
}


void initialize_u (char *pathname)
{
        unsigned int m,t;

        strcpy(fin[6],"u.dat");
        strcpy(filename,pathname);;
        strcat(filename,fin[6]);
        if ((pin[6]=fopen(filename,"wt"))==NULL)
            {
                fprintf(flog,"Unable to open input file=%s",filename);
                exit(1);
            }
        for (m=1;m<=I; m++)
        {
```

```
            for (t=1; t<=T; t++)
                fprintf(pin[6],"0 ");
            fprintf(pin[6],"\n");
        }
        fclose(pin[6]);
}

void initialize_u3(void)
{
        unsigned int m,t;

        u3=(float **)alloc_matrix(I+1,T+1,sizeof(float));
        for (m=0;m<=I+1;m++)
            for (t=0;t<=T+1;t++)
                u3[m][t]=0.0;
}

void store_u3(void)
{
        unsigned int m,j;

        for (m=0;m<=I;m++)
            for (j=0;j<=T;j++)
                u3[m][j]=u[m][j];
}

void copyuu3(void)
{
        unsigned int m,j;

        u=(float **)alloc_matrix(I+1,T+1,sizeof(float));
        for (m=0;m<=I;m++)
            for (j=0;j<=T;j++)
                u[m][j]=u3[m][j];
}
```

APPENDIX C

DETABA AND TANE CODES

```
// DETABA code
// program to perform Level2 of Hierarchical Decomposition
// program to perform single depot VRP for each depot and time
// include TANE

#include <bios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloc.h>
#include <time.h>
#include <conio.h>
#include <math.h>
#include <values.h>


// Function declarations
void *alloc_vector(unsigned int elements,size_t size);
void **alloc_matrix(unsigned int rows,unsigned int columns,size_t size);
void free_matrix(float **m,unsigned int rows);
void free_matrix2(unsigned int **m,unsigned int rows);
void free_matrix21(unsigned int **m,unsigned int rows);
void free_matrix3(long **m,unsigned int rows);
void read_command_file(char *pathname);
void num2str(int num,char *str);
void get_files(void);
void skip_line(FILE *file);
void open_input_files(char *pathname);
void initialize(void);
void calculate_dc(void);
void prepare_liste(void);
void calculate_inlen(void);
void prepare_com01_distr(void);
void prepare_output_files(char *pathname);
void free_arrays(void);
void free_arrays_op(void);
void free_arrays_end(void);
void ***d3alloc(unsigned int depth);
void **d2alloc(unsigned int rows);

void arraycpy(unsigned int *gec1,unsigned int *gec2, unsigned int elements);
void arraycpy2(float *gec1,float *gec2, unsigned int elements);
void arraycpy3(long *gec1,long *gec2, unsigned int elements);
void arrayred(unsigned int *gec1, unsigned int point, unsigned int elements);
void arrayredf(float *gec1, unsigned int point, unsigned int elements);
void arrayredl(long *gec1, unsigned int point, unsigned int elements);
void matrixcpy(unsigned int **gec1,unsigned int **gec2);
void initial_vector(unsigned int *gec1,unsigned int elements);
void initial_vector2(float *gec1,unsigned int elements);
void initial_vector3(long *gec1,unsigned int elements);
void initial_matrix(unsigned int **gec1,unsigned int rows,unsigned int columns);
void initial_matrix2(float **gec1,unsigned int rows,unsigned int columns);

void choose_routes_vertices(void);
void insert(unsigned int GR,unsigned int *gec, unsigned int elements, unsigned int *vv, unsigned int vertices,
float *routcost);
void twoopt(unsigned int *gec, unsigned int elements, float *routcost);
void tabu_check(unsigned int *gec,unsigned int r,unsigned int route);
```

```
void search_neighborhood(void);
void update_best_neighbor(void);
void update_tabu_list(void);
void update_route_list(void);
void route_up(unsigned int nn);
void check_empty_routes(void);
void initialize_neighborhood_search(void);
void neighborhood_search(void);
void initial_solution(void);
void display_input(void);
void display(unsigned int GR,unsigned int *gec, unsigned int elements, float *routcost);
void display_route(unsigned int **gec1,long *gec2,float *gec3,float Ftot);
void display_tabu(void);
void matrix_allocation(unsigned int **gec,unsigned int rows,unsigned int columns);

void allocate_initial(void);
void generate_initial_solution(void);
void first_improvement(void);
void check_best_initial(unsigned int *e);
void allocate_startingpoint(void);
void prepare_initial(void);

void matrixcpyRopR(void);
void matrixcpyRiRop(void);
void matrixcpyRRi(void);

void solution_improvement(void);
void allocate_op(void);
void keep_solution(void);
void il_allocation(void);
void neighborhood_search2(void);
void display_il(void);

void intensification(void);
void generate_intensification_solution(unsigned int *e);
void intensify(void);
void twoopt2(unsigned int *gec, unsigned int elements, float *routcost);

#define itlim 20
#define M 2
#define P 2
#define vmax 201

// Global variables
unsigned int I;          // Number of items
unsigned int K;          // Number of customers
unsigned int T;          // Production horizon
unsigned int J;          // Number of depots

unsigned int i,k,j,t;
unsigned int R1,R2,r1,r2;
unsigned int N,V;
unsigned int mu,pi;
unsigned int *R[vmax];
long cap, *d;
float *F;
long *slack;
```

```
float *x, *y, **dc;
unsigned int *v1, *v2;

unsigned int *Rop[vmax];
float *Fop;
long *Sop;

long change1,change2;
unsigned int *G1, *G2;
unsigned int **tv, **tr;
unsigned int tabu;
unsigned int feasible;
float Fmin,Ftmin,GF1,GF2,Fprev,Fopt,Finit,Fp,Fcum;
unsigned int RR1,RR2, *GG1, *GG2, *t1, *t2;
float GGF1, GGF2;
long slack1, slack2;

char fin[1][13];                    // Input data file names
FILE *pin[1];                       // Input data file pointers
char fout[5][13];                   // Output data file names
FILE *pout[5];                      // Output data file pointers

char filename[100];                 // general file name with its path
char dirname[100];
char com[13]={"com02.dat"};         // name of command file
char logfile[13]={"com11.log"};     // name of log file

FILE *cfile;                        // Command file pointer
FILE *flog;                         // Log file pointer

unsigned int p,pop,nopt,it,itbest;

unsigned int *Ri[vmax];
float *Fi,Fin;
long *Si;

unsigned int alfa,gama;
unsigned int Qlow,Qup,Q;

unsigned int *liste,inlen,rr,rrr,kk,mlow;

unsigned int ***RR;
float **FF;
long **SS;
unsigned int *il;
unsigned int neigbul;


void main(int argc,char *argv[])
{
    clock_t start,end;
    // Clear the screen
    clrscr();
    start=clock();
//  randomize();
    // Command line check
    if(argc!=2)
    {
```

```
        puts("argument count mismatch");
        exit(1);
    }
    // Open log file
    strcpy(filename,argv[1]);
    strcat(filename,logfile);
    if((flog=fopen(filename,"wt"))==NULL)
    {
        puts("Unable to open log file.");
        exit(1);
    }
    // Read command file
    strcpy(dirname,argv[1]);
    open_input_files(dirname);
    Fcum=0;
    for (j=1;j<=J;j++)
        for (t=1;t<=T;t++)
        {
//          fprintf(flog,"\n(%u,%u)",j,t);
            read_command_file(argv[1]);
            initialize();
            if (N>=2)
            {
            calculate_dc();
            prepare_liste();
            calculate_inlen();
//          display_input();
            initial_solution();
//          il_allocation();
            solution_improvement();
//          intensification();
            free_arrays();
            free_matrix(dc,N+1);
            free(d);
    //      free(il);
            }
            else if (N==1)
            {
            calculate_dc();
            Fopt=dc[0][1]+dc[1][0];
            free_matrix(dc,N+1);
            }
            else
            Fopt=0;
            fprintf(flog,"\nF(%u,%u)=%-15.2f",j,t,Fopt);
            Fcum+=Fopt;
        }
    fprintf(flog,"\nFproblem=%-15.2f",Fcum);
    end=clock();
    fprintf(flog,"\nCPU TIME = %f ",(end-start)/CLK_TCK);
    fclose(pin[0]);
    fclose(cfile);
    fclose(flog);
}


// Function to allocate memory for vector
void *alloc_vector(unsigned int elements,size_t size)
```

```
{
      void *temp;
      if((temp=malloc(elements*size))==NULL)
      {
            fputs("Unable to allocate memory.",flog);
            exit(1);
      }
      return(temp);
}

      // Function to allocate memory for matrix
void **alloc_matrix(unsigned int rows,unsigned int columns,size_t size)
{
      void **a;
      unsigned int i;
      if((a=malloc(rows*sizeof(void *)))==NULL)
      {
            fputs("Memory Allocation Error!",flog);
            exit(1);
      }
      for(i=0;i<rows;i++)
      {
            if((*(a+i)=malloc(columns*size))==NULL)
            {
                  fputs("Memory Allocation Error!",flog);
                  exit(1);
            }
      }
      return(a);
}


void ***d3alloc(unsigned int depth)
{
      void ***a;

      if((a=malloc(depth*sizeof(void **)))==NULL)
      {
            fputs("Memory Allocation Error!",flog);
            exit(1);
      }
      return(a);
}


void **d2alloc(unsigned int rows)
{
      void **a;

      if((a=malloc(rows*sizeof(void *)))==NULL)
      {
            fputs("Memory Allocation Error!",flog);
            exit(1);
      }
      return(a);

}
```

```
// Free previously allocated matrix memory for float values
void free_matrix(float **m,unsigned int rows)
{
	unsigned int i;
	for(i=0;i<rows;i++)free(*(m+i));
	free(m);
}

// Free previously allocated matrix memory for integer values
void free_matrix2(unsigned int **m,unsigned int rows)
{
	unsigned int i;
	for(i=0;i<rows;i++)free(*(m+i));
//	free(m);
}

void free_matrix21(unsigned int **m,unsigned int rows)
{
	unsigned int i;
	for(i=0;i<rows;i++)free(*(m+i));
	free(m);
}

void free_matrix3(long **m,unsigned int rows)
{
	unsigned int i;
	for(i=0;i<rows;i++)free(*(m+i));
	free(m);
}

void arraycpy(unsigned int *gec1,unsigned int *gec2, unsigned int elements)
{
	unsigned int b;
	for (b=0;b<=elements;b++)
	{
		gec1[b]=gec2[b];
	}
}


void arraycpy2(float *gec1,float *gec2, unsigned int elements)
{
	unsigned int b;
	for (b=0;b<=elements;b++)
	{
		gec1[b]=gec2[b];
	}
}

void arraycpy3(long *gec1,long *gec2, unsigned int elements)
{
	unsigned int b;
	for (b=0;b<=elements;b++)
	{
		gec1[b]=gec2[b];
	}
}
```

```
void arrayred(unsigned int *gec1, unsigned int point, unsigned int elements)
{
    unsigned int b;
    for (b=point;b<elements;b++)
    {
        gec1[b]=gec1[b+1];
    }
    gec1[elements]=0;
}

void arrayredf(float *gec1, unsigned int point, unsigned int elements)
{
    unsigned int b;
    for (b=point;b<elements;b++)
    {
        gec1[b]=gec1[b+1];
    }
    gec1[elements]=0;
}

void arrayredl(long *gec1, unsigned int point, unsigned int elements)
{
    unsigned int b;
    for (b=point;b<elements;b++)
    {
        gec1[b]=gec1[b+1];
    }
    gec1[elements]=cap;
}

void matrixcpy(unsigned int **gec1,unsigned int **gec2)
{
    unsigned int i,b,a;

    a=gec1[0][0];
    free_matrix2(gec1,a+1);
    a=gec2[0][0];
    matrix_allocation(gec1,a+1,N+1);
    for (b=0;b<=a;b++)
        for (i=0;i<=N;i++)
            gec1[b][i]=gec2[b][i];
}


void initial_vector(unsigned int *gec1,unsigned int elements)
{
    unsigned int i;
    for (i=0;i<=elements;i++)
        gec1[i]=0;
}


void initial_vector2(float *gec1,unsigned int elements)
{
    unsigned int i;
    for (i=0;i<=elements;i++)
        gec1[i]=0;
}
```

```
void initial_vector3(long *gec1,unsigned int elements)
{
     unsigned int i;
     for (i=0;i<=elements;i++)
          gec1[i]=0;
}


void initial_matrix(unsigned int **gec1,unsigned int rows,unsigned int columns)
{
     unsigned int i,m;
     for (i=0;i<=rows;i++)
          for (m=0;m<=columns;m++)
               gec1[i][m]=0;
}


void initial_matrix2(float **gec1,unsigned int rows,unsigned int columns)
{
     unsigned int i,m;
     for (i=0;i<=rows;i++)
          for (m=0;m<=columns;m++)
               gec1[i][m]=0;
}


// Function to read characters until new line from file
void skip_line(FILE *file)
{
     char ch;
     do   ch=fgetc(file); while(ch!=10);
}

// Function to open and process command file
void read_command_file(char *pathname)
{

     // Read contents of command file
     skip_line(cfile);
     skip_line(cfile);
     fscanf(cfile,"%u",&N);
///  fprintf(flog,"Number of customers : %u\n",N);
     fgetc(cfile);

     skip_line(cfile);
     fscanf(cfile,"%ld",&cap);
///  fprintf(flog,"Vehicle Capacity : %ld\n",cap);

}

void num2str(int num,char *str)
{
     int hundt=0,ten=0,one=0;

     while (num>=100)
     {
          num=num-100;
```

```
            hundt++;
        }
        while (num>=10)
        {
            num=num-10;
            ten++;
        }
        one=num;

        str[0]=hundt+48;    //to convert char
        str[1]=ten+48;
        str[2]=one+48;
        str[3]=NULL;
}

// Function to open input files
void open_input_files(char *pathname)
{
        fputs("Processing command file...\n\n",flog);

        strcpy(filename,pathname);
        fprintf(flog,"Network directory: %s\n",filename);
            strcpy(fin[0],"input.dat");
            strcat(filename,fin[0]);
            // Open file
            if((pin[0]=fopen(filename,"rt"))==NULL)
            {
                fprintf(flog,"Unable to open input file= %s",filename);
                exit(1);
            }
        skip_line(pin[0]);
        skip_line(pin[0]);
        skip_line(pin[0]);

        skip_line(pin[0]);
        fscanf(pin[0],"%u",&I);
        fprintf(flog,"Number of items :\n%u",I);
        fgetc(pin[0]);

        skip_line(pin[0]);
        fscanf(pin[0],"%u",&K);
        fprintf(flog,"\nNumber of customers :\n%u",K);
        fgetc(pin[0]);

        skip_line(pin[0]);
        fscanf(pin[0],"%u",&T);
        fprintf(flog,"\nNumber of periods :\n%u",T);
        fgetc(pin[0]);

        skip_line(pin[0]);
        fscanf(pin[0],"%u",&J);
        fprintf(flog,"\nNumber of depots:\n%u",J);

        // Open command file
        strcpy(filename,pathname);
        strcat(filename,com);
        if((cfile=fopen(filename,"rt"))==NULL)
        {
```

```
            fputs("Unable to open command file.",flog);
            exit(1);
        }
}

// Function to initialize operation arrays
void initialize(void)
{
    unsigned int b;                                    // Counter variable

    // Allocate memory for input var. matrices
    x=(float *)alloc_vector(N+1,sizeof(float));
    y=(float *)alloc_vector(N+1,sizeof(float));
    d=(long *)alloc_vector(N+1,sizeof(long));

    // Initialize all arrays and matrices with zero values
    initial_vector2(x,N);
    initial_vector2(y,N);
    initial_vector3(d,N);

    //Load input data
    skip_line(pin[0]);
    skip_line(pin[0]);
    fscanf(pin[0],"%f",&x[0]);
    fscanf(pin[0],"%f",&y[0]);
    for(b=1;b<=N;b++)
    {
        fscanf(pin[0],"%f",&x[b]);
        fscanf(pin[0],"%f",&y[b]);
        fscanf(pin[0],"%ld",&d[b]);
///     d[b]=50*d[b];
    }
}

void calculate_dc(void)
{
    unsigned int b,c;

    dc=(float **)alloc_matrix(N+1,N+1,sizeof(float));
    initial_matrix2(dc,N,N);

    for (b=0;b<=N;b++)
        for (c=b+1;c<=N;c++)
        {
            dc[b][c]=(x[b]-x[c])*(x[b]-x[c]);
            dc[b][c]+=(y[b]-y[c])*(y[b]-y[c]);
            dc[b][c]=100000*sqrt(dc[b][c]);
            dc[c][b]=dc[b][c];
        }
    free(x);
    free(y);
}


// Function to free memory allocated for arrays and matrices
void free_arrays(void)
{
    unsigned int n;
```

```
        free(F);
        free(slack);
        n=R[0][0];
        free_matrix2(R,n+1);
        free_matrix21(tv,Q+1);
        free_matrix21(tr,Q+1);

}

void free_arrays_op(void)
{
        unsigned int n;

        free(Fop);
        free(Sop);
        n=Rop[0][0];
        free_matrix2(Rop,n+1);
}

void free_arrays_end(void)
{
        unsigned int n;

        free(F);
        free(slack);
        n=R[0][0];
        free_matrix2(R,n+1);

}

void choose_routes_vertices(void)
{
        unsigned int xx,x1,y1,number,itr,iter,m,same;
        float temp;
        itr=1;
        do
        {
                feasible=0;
                do
                {
                        x1=R[0][0];
                        R1=(rand()%x1)+1;
                        do
                        {
                                R2=(rand()%x1)+1;
                        }
                        while (R1==R2);
                }
                while ((R[0][R1]==0)&&(R[0][R2]==0));
                m=R[0][R1];
                G1=(unsigned int *)alloc_vector(m+3,sizeof(int));
                initial_vector(G1,m+2);
                m=R[0][R2];
                G2=(unsigned int *)alloc_vector(m+3,sizeof(int));
                initial_vector(G2,m+2);
                x1=M>R[0][R1]?R[0][R1]:M;
                y1=P>R[0][R2]?R[0][R2]:P;
```

```
iter=1;
do
{
     same=0;
     change1=change2=0;
     if (y1==0)
          mu=(rand()%x1)+1;
     else
          mu=rand()%(x1+1);
     if (mu==0)
          pi=(rand()%y1)+1;
     else
          pi=rand()%(y1+1);
     if ((mu==R[0][R1])&&(pi==R[0][R2]))
          same=1;
     if (same==0)
     {
     v1=(unsigned int *)alloc_vector(M+1,sizeof(int));
     initial_vector(v1,M);
     v2=(unsigned int *)alloc_vector(P+1,sizeof(int));
     initial_vector(v2,P);
     arraycpy(G1,R[R1],R[0][R1]);
     GF1=F[R1];
     number=R[0][R1];
     for (m=1;m<=mu;m++)
     {
          xx=(rand()%number)+1;
          GF1+=dc[G1[xx-1]][G1[xx+1]]-(dc[G1[xx-1]][G1[xx]]+dc[G1[xx]][G1[xx+1]]);
          v1[m]=G1[xx];
          change1+=d[v1[m]];
          change2-=d[v1[m]];
          arrayred(G1,xx,number);
          number--;
     }
     if (R[0][R1]==mu)
          GF1=0;
     arraycpy(G2,R[R2],R[0][R2]);
     GF2=F[R2];
     number=R[0][R2];
     for (m=1;m<=pi;m++)
     {
          xx=(rand()%number)+1;
          temp=dc[G2[xx-1]][G2[xx+1]]-(dc[G2[xx-1]][G2[xx]]+dc[G2[xx]][G2[xx+1]]);
          GF2=GF2+temp;
          v2[m]=G2[xx];
          change2+=d[v2[m]];
          change1-=d[v2[m]];
          arrayred(G2,xx,number);
          number--;
     }
     if (R[0][R2]==pi)
          GF2=0;
     change1+=slack[R1];
     change2+=slack[R2];
     if ((change1>=0)&&(change2>=0))
          feasible=1;
     if (feasible!=1)
     {
```

```
                        inscost=newcost;
                        e=a;
                    }
                    index=nextindex;
                }
                for (a=elements; a>=e; a--)
                    gec[a+1]=gec[a];
                gec[e]=nearest;
            }
            else
            {
                gec[1]=nearest;
                inscost=dc[0][nearest]+dc[nearest][0];
            }
            elements++;
/**         fprintf(flog,"\n");
            for (a=0;a<=elements;a++)
                fprintf(flog,"%u ",gec[a]);
            fprintf(flog,"0 "); **/
            arrayred(v,vertexno,b);
            b--;
            *routcost+=inscost;
///         fprintf(flog,"/Routcost=%-8.2f",*routcost);
            free(dist);
        }
        free(v);
}

void twoopt(unsigned int *gec, unsigned int elements, float *routcost)
{
    unsigned int a,itr,I1,I2,J1,J2,next,last,ahead,index,Bmax;
    unsigned int *ptr;
    float max1;

    Bmax=(elements+1)*(elements-2)/4;
    ptr=(unsigned int *)alloc_vector(N+1,sizeof(int));
    initial_vector(ptr,N);
    for (a=0;a<elements;a++)
        ptr[gec[a]]=gec[a+1];
    ptr[gec[elements]]=0;
    itr=1;
    do
    {
        a=rand()%(elements+1);
        I1=gec[a];
        I2=ptr[I1];
        J1=rand()%(elements-2);
        if (a==0)
            J1+=2;
        else if (a==elements)
            J1++;
        else if (J1>=a-1)
            J1+=3;
        J1=gec[J1];
        J2=ptr[J1];
///     fprintf(flog,"\nitr=%u  ",itr);
///     fprintf(flog,"twoopt arcs=(%u,%u)-(%u,%u), ",I1,I2,J1,J2);
        max1=dc[I1][I2]+dc[J1][J2]-(dc[I1][J1]+dc[I2][J2]);
```

```
                    free(v1);
                    free(v2);
                }
                }
                iter++;
            }
            while ((feasible==0)&&(iter<=itlim));
            if (feasible!=1)
            {
                free(G1);
                free(G2);
            }
        itr++;
        }
        while ((feasible==0)&&(itr<=itlim));
}

void insert(unsigned int GR,unsigned int *gec, unsigned int elements, unsigned int *vv, unsigned int vertices,
float *routcost)
{
    unsigned int b,c,a,vertexno,nearest,index,nextindex,e;
    float mindist, *dist, inscost, newcost;
    unsigned int *v;

    v=(unsigned int *)alloc_vector(vertices+1,sizeof(int));
    arraycpy(v,vv,vertices);
    b=vertices;
    while (b>0)
    {
        mindist=MAXFLOAT;
        dist=(float *)alloc_vector(b+1,sizeof(float));
        for (c=0;c<=b;c++)
         dist[c]=MAXFLOAT;
        for (c=1;c<=b;c++)
        {
            for (a=0;a<=elements;a++)
            {
                if (dc[gec[a]][v[c]]<dist[c])
                    dist[c]=dc[gec[a]][v[c]];
            }
            if (dist[c]<mindist)
            {
                vertexno=c;
                mindist=dist[c];
            }
        }
        nearest=v[vertexno];
///     fprintf(flog,"\ninsert %u into R=%u",nearest,GR);
        inscost=MAXFLOAT;
        if (elements!=0)
        {
            index=0;
            for (a=1;a<=elements+1;a++)
            {
                nextindex=gec[a];
                newcost=dc[index][nearest]+dc[nearest][nextindex]-dc[index][nextindex];
                if (newcost<inscost)
                {
```

```
///         fprintf(flog," cost saving=%-8.2f",max1);
            if (max1>0)
            {
                ptr[I1]=J1;
                next=I2;
                last=J2;
                do
                {
                    ahead=ptr[next];
                    ptr[next]=last;
                    last=next;
                    next=ahead;
                }
                while (next!=J2);
                index=0;
                for (a=0;a<=elements;a++)
                {
                    gec[a]=index;
                    index=ptr[index];
                }
                *routcost-=max1;
            }
            itr++;
        }
        while (itr<=Bmax);

        free(ptr);
}

void tabu_check(unsigned int *gec,unsigned int r,unsigned int route)
{
        unsigned int m,n,c;

        tabu=0;
        m=1;
        while ((m<=r)&&(tabu==0))
        {
            for (n=1;n<=Q;n++)
                for (c=1;c<=tv[n][0];c++)
                    if ((tv[n][c]==gec[m])&&(tr[n][c]==route))
                        tabu=1;
            m++;
        }
}

void search_neighborhood(void)
{
        float Ftemp;
        unsigned int found;

        found=0;
        Ftemp=GF1+GF2+Fprev-F[R1]-F[R2];
///     fprintf(flog,"\nFneighbor=%-8.2f",Ftemp);
            tabu_check(v1,mu,R2);
            if (tabu==0)
                tabu_check(v2,pi,R1);
            if ((tabu==1)&&(Ftemp<Fopt))
                found=1;
```

```
        else if ((tabu==0)&&(Ftemp<Fmin))
            found=1;
        if (found==1)
        {
            neigbul=1;
            Fmin=Ftemp;
            itbest=it;
            update_best_neighbor();
        }
        else if (neigbul==0)
        {
            Ftmin=Ftemp;
            itbest=it;
            update_best_neighbor();
        }

}

void update_best_neighbor(void)
{
    unsigned int a;

    RR1=R1;
    RR2=R2;
    r1=R[0][R1]-mu+pi;
    free(GG1);
    GG1=(unsigned int *)alloc_vector(r1+1,sizeof(int));
    arraycpy(GG1,G1,r1);
    r2=R[0][R2]-pi+mu;
    free(GG2);
    GG2=(unsigned int *)alloc_vector(r2+1,sizeof(int));
    arraycpy(GG2,G2,r2);
    slack1=change1;
    slack2=change2;
    GGF1=GF1;
    GGF2=GF2;
    arraycpy(t1,v1,mu);
    for (a=mu+1;a<=M;a++)
        t1[a]=0;
    t1[M+1]=mu;
    arraycpy(t2,v2,pi);
    for (a=pi+1;a<=P;a++)
        t2[a]=0;
    t2[P+1]=pi;
}


void update_tabu_list(void)
{
    unsigned int c,m,n;

    if (tv[0][0]<Q)
        tv[0][0]++;
    for (n=tv[0][0];n>1;n--)
    {
        for (m=0;m<=tv[n-1][0];m++)
        {
            tv[n][m]=tv[n-1][m];
```

```
                tr[n][m]=tr[n-1][m];
            }
            for (m=tv[n-1][0]+1;m<=tv[n][0];m++)
            {
                tv[n][m]=0;
                tr[n][m]=0;
            }
        }
        initial_vector(tv[1],M+P);
        initial_vector(tr[1],M+P);
        for (c=1;c<=t1[M+1];c++)
        {
            tr[1][c]=RR1;
            tv[1][c]=t1[c];
            tv[1][0]++;
        }
        n=t1[M+1];
        m=t2[P+1];
        for (c=n+1;c<=n+m;c++)
        {
            tr[1][c]=RR2;
            tv[1][c]=t2[c-n];
            tv[1][0]++;
        }
}


void update_route_list(void)
{
    unsigned int n;

    arraycpy(R[RR1],GG1,r1);
    for (n=r1+1;n<=R[0][RR1];n++)
        R[RR1][n]=0;
    arraycpy(R[RR2],GG2,r2);
    for (n=r2+1;n<=R[0][RR2];n++)
        R[RR2][n]=0;
    R[0][RR1]=r1;
    R[0][RR2]=r2;
    slack[RR1]=slack1;
    slack[RR2]=slack2;
    F[RR1]=GGF1;
    F[RR2]=GGF2;
}

void check_empty_routes(void)
{
    unsigned int n,m,a,c;

    if ((R[0][RR1]==0)||(R[0][RR2]==0))
    {
        m=R[0][RR1]==0?RR1:RR2;
        n=R[0][0];
        arrayredf(F,m,n);
        arrayredl(slack,m,n);
        for (n=m;n<R[0][0];n++)
        {
            for (c=0;c<=R[0][n+1];c++)
```

```
                R[n][c]=R[n+1][c];
            for (c=R[0][n+1]+1;c<=R[0][n];c++)
                R[n][c]=0;
            R[0][n]=R[0][n+1];
        }
        initial_vector(R[n],N);
        R[0][n]=0;
        for (a=m;a<R[0][0];a++)
            for (n=1;n<=Q;n++)
                for (c=1;c<=tv[n][0];c++)
                    if (tr[n][c]==a+1)
                        tr[n][c]=a;
        n=R[0][0];
        if (R[0][n-1]==0)
        {
            R[0][0]--;
            free(R[n]);
        }
    }
    else
    {
        m=R[0][0];
        if ((R[0][m]>0)&&(m<N))
        {
            m++;
            R[m]=(unsigned int *)alloc_vector(N+1,sizeof(int));
            initial_vector(R[m],N);
            R[0][0]++;
            R[0][m]=0;
        }
    }
}

void initialize_neighborhood_search(void)
{

    GG1=(unsigned int *)alloc_vector(N+2,sizeof(int));
    GG2=(unsigned int *)alloc_vector(N+2,sizeof(int));
//  initial_vector(GG1,N+1);
//  initial_vector(GG2,N+1);
    t1=(unsigned int *)alloc_vector(M+2,sizeof(int));
    initial_vector(t1,M+1);
    t2=(unsigned int *)alloc_vector(P+2,sizeof(int));
    initial_vector(t2,P+1);
    RR1=RR2=0;
    slack1=slack2=0;
    GGF1=GGF2=0;
    r1=r2=0;
    Fmin=Ftmin=MAXFLOAT;
    neigbul=0;
    itbest=0;
}

void neighborhood_search(void)
{
    unsigned int number,a,m,n;

    initialize_neighborhood_search();
```

```
        for (it=1;it<=gama;it++)
        {
            choose_routes_vertices();
            if (feasible==1)
            {
 /**            fprintf(flog,"\n\nNeighbor%u:",it);
                fprintf(flog,"\nR1=%u/",R1);
                for(a=1;a<=mu;a++)
                    fprintf(flog,"%u,",v1[a]);
                fprintf(flog," mu=%u ",mu);
                fprintf(flog," GF1=%-8.2f",GF1);
                fprintf(flog,"\nR2=%u/",R2);
                for(a=1;a<=pi;a++)
                    fprintf(flog,"%u,",v2[a]);
                fprintf(flog," pi=%u",pi);
                fprintf(flog," GF2=%-8.2f",GF2); **/
                number=R[0][R1]-mu;
                insert(R1,G1,number,v2,pi,&GF1);
                number=R[0][R1]-mu+pi;
 ///            display(R1,G1,number,&GF1);
                if (number>=3)
                    twoopt2(G1,number,&GF1);
                number=R[0][R2]-pi;
                insert(R2,G2,number,v1,mu,&GF2);
                number=R[0][R2]-pi+mu;
 ///            display(R2,G2,number,&GF2);
                if (number>=3)
                    twoopt2(G2,number,&GF2);
                search_neighborhood();
                free(G1);
                free(G2);
                free(v1);
                free(v2);
            }
        }
        if (feasible==1)
        {
            if (neigbul==0)
                Fmin=Ftmin;
            update_route_list();
            check_empty_routes();
            update_tabu_list();
            Fprev=Fmin;
            if (Fmin<Fopt)
            {
                Fopt=Fmin;
                matrixcpyRopR();
                arraycpy2(Fop,F,N);
                arraycpy3(Sop,slack,N);
                pop=p;
            }
        }
 /**    fprintf(flog,"\n\nNeighbor Search Results:");
        fprintf(flog,"\nBest neighbor=Neighbor%u",itbest);
        fprintf(flog,"\nTotal Routes:%u",R[0][0]);
        fprintf(flog,"\nRoute Matrix\n");
        for(m=1;m<=R[0][0];m++)
        {
```

```
                for(n=0;n<=R[0][m];n++)
                {
                    fprintf(flog,"%u ",R[m][n]);
                }
                fprintf(flog,"0      /");
                fprintf(flog,"slack=%ld  /Froute=%-8.2f\n",slack[m],F[m]);
            }  **/
///         fprintf(flog,"\n%-5u %15.2f",p,Fprev);
///         display_tabu();
///     fprintf(flog,"%15.2f %-6u",Fopt,pop);
        free(GG1);
        free(GG2);
        free(t1);
        free(t2);
}

void initial_solution(void)
{
    unsigned int numin,jj;

    Qlow=5;
    Qup=10;
    Q=rand()%(Qup-Qlow+1);
    Q=Qlow+Q;
    Q=N/2;
    numin=ceil(sqrt(N)/2);
    allocate_initial();
    for (jj=1;jj<=numin;jj++)
    {
///         fprintf(flog,"\n*****Initial solution=%u******",jj);
        generate_initial_solution();
        first_improvement();
        check_best_initial(&jj);
        free_arrays();
        free_arrays_op();
    }
    allocate_startingpoint();
    prepare_initial();

///   fprintf(flog,"\n\n**********BEST STARTING SOLUTION**********");
///   display_route(R,slack,F,Fprev);
    free(liste);
}

void allocate_initial(void)
{
    unsigned int a;

    Ri[0]=(unsigned int *)alloc_vector(N+1,sizeof(int));
    Fi=(float *)alloc_vector(N+1,sizeof(float));
    Si=(long *)alloc_vector(N+1,sizeof(long));
    for (a=0;a<=N;a++)
        Si[a]=cap;
    initial_vector2(Fi,N);
    initial_vector(Ri[0],N);
    Fin=MAXFLOAT;
    nopt=0;
}
```

```
void generate_initial_solution(void)
{
    unsigned int x1,y1,m,n,a,k;

    R[0]=(unsigned int *)alloc_vector(N+1,sizeof(int));
    R[1]=(unsigned int *)alloc_vector(N+1,sizeof(int));
    tv=(unsigned int **)alloc_matrix(Q+1,M+P+1,sizeof(int));
    tr=(unsigned int **)alloc_matrix(Q+1,M+P+1,sizeof(int));
    initial_matrix(tr,Q,M+P);
    initial_matrix(tv,Q,M+P);
    F=(float *)alloc_vector(N+1,sizeof(float));
    slack=(long *)alloc_vector(N+1,sizeof(long));
    for (a=0;a<=N;a++)
        slack[a]=cap;
    initial_vector2(F,N);
    initial_vector(R[0],N);
    initial_vector(R[1],N);
    Fprev=0;
    y1=(rand()%liste[0])+1;
    x1=liste[y1];
    arrayred(liste,y1,N);
    liste[0]--;
    m=1;
    n=1;
    for (a=x1;a<=N;a++)
    {
        if (slack[m]<d[a])
        {
            m++;
            R[m]=(unsigned int *)alloc_vector(N+1,sizeof(int));
            initial_vector(R[m],N);
            n=1;
        }
        slack[m]-=d[a];
        R[m][n]=a;
        n++;
        R[0][m]++;
    }
    for (a=1;a<=x1-1;a++)
    {
        if (slack[m]<d[a])
        {
            m++;
            R[m]=(unsigned int *)alloc_vector(N+1,sizeof(int));
            initial_vector(R[m],N);
            n=1;
        }
        slack[m]-=d[a];
        R[m][n]=a;
        n++;
        R[0][m]++;
    }
    if (m<N)
    {
        R[0][0]=m+1;
        R[m+1]=(unsigned int *)alloc_vector(N+1,sizeof(int));
        initial_vector(R[m+1],N);
```

```
    }
    else
        R[0][0]=m;
    for (m=1;m<=R[0][0];m++)
    {
        for (a=0;a<R[0][m];a++)
            F[m]+=dc[R[m][a]][R[m][a+1]];
        F[m]+=dc[R[m][R[0][m]]][0];
        Fprev+=F[m];
    }
///  display_route(R,slack,F,Fprev);
    Rop[0]=(unsigned int *)alloc_vector(N+1,sizeof(int));
    initial_vector(Rop[0],N);
    Fop=(float *)alloc_vector(N+1,sizeof(float));
    Sop=(long *)alloc_vector(N+1,sizeof(long));
    matrixcpyRopR();
    arraycpy2(Fop,F,N);
    arraycpy3(Sop,slack,N);
    Fopt=Fprev;
}

void first_improvement(void)
{

    alfa=N;
    gama=5;
    for (p=1;p<=alfa;p++)
    {
///     fprintf(flog,"\n\nIteration=%u",p);
        neighborhood_search();
    }
}


void check_best_initial(unsigned int *e)
{
    unsigned int n;

    if (Fopt<Fin)
    {
        nopt=*e;
        arraycpy2(Fi,Fop,Rop[0][0]);
        arraycpy3(Si,Sop,Rop[0][0]);
        for (n=Rop[0][0]+1;n<=Ri[0][0];n++)
        {
            Fi[n]=0;
            Si[n]=0;
        }
        matrixcpyRiRop();
        Fin=Fopt;
    }
///  fprintf(flog,"%-15.2f %-6u",Fin,nopt);
}

void allocate_startingpoint(void)
{
    R[0]=(unsigned int *)alloc_vector(N+1,sizeof(int));
    initial_vector(R[0],N);
```

```
        F=(float *)alloc_vector(N+1,sizeof(float));
        slack=(long *)alloc_vector(N+1,sizeof(long));

}

void prepare_initial(void)
{
        unsigned int n;
        matrixcpyRRi();
        arraycpy2(F,Fi,N);
        arraycpy3(slack,Si,N);
        free(Si);
        free(Fi);
        n=Ri[0][0];
        free_matrix2(Ri,n+1);
        Fprev=Fin;
}

void display_input(void)
{
        unsigned int i,a;

        fprintf(flog,"\nDEMAND AND DISTANCE MATRIX");
        fprintf(flog,"\n");
        fprintf(flog,"   ");
        fprintf(flog,"q     ");
        for (i=0;i<=N;i++)
                fprintf(flog,"%-7u",i);
        for (i=1;i<=N;i++)
        {
                fprintf(flog,"\n");
                fprintf(flog,"%-3u",i);
                fprintf(flog,"%-6ld",d[i]);
                for (a=0;a<=i-1;a++)
                        fprintf(flog,"%-7.2f",dc[i][a]);
        }
}

void display_route(unsigned int **gec1,long *gec2,float *gec3,float Ftot)
{
        unsigned int m,n;

        fprintf(flog,"\nRoute Matrix\n");
        for (m=1;m<=gec1[0][0];m++)
        {
                for (n=0;n<=gec1[0][m];n++)
                        fprintf(flog,"%u ",gec1[m][n]);
                fprintf(flog,"0 /");
                fprintf(flog,"slack=%ld /Froute=%-15.2f\n",gec2[m],gec3[m]);
        }
        fprintf(flog,"Ftotal=%-15.2f",Ftot);
}


void display(unsigned int GR,unsigned int *gec, unsigned int elements, float *routcost)
{
        unsigned int a;
```

```
    fprintf(flog,"\nR=%u /",GR);
    for (a=0;a<=elements;a++)
        fprintf(flog,"%u ",gec[a]);
    fprintf(flog,"0/ Routecost=%-8.2f",*routcost);
}

void matrix_allocation(unsigned int **gec,unsigned int rows,unsigned int columns)
{
    unsigned int a;

    for(a=0;a<rows;a++)
        gec[a]=(unsigned int *)alloc_vector(columns,sizeof(int));
}

void display_tabu(void)
{
    unsigned int c,m;

    fprintf(flog,"\nTABU LIST");
    for (c=1;c<=Q;c++)
    {
        fprintf(flog,"\n");
        for (m=1;m<=tv[c][0];m++)
            fprintf(flog,"(%u/%u),",tr[c][m],tv[c][m]);
    }
}

void route_up(unsigned int nn)
{
    unsigned int i;

    for (i=0;i<=R[0][nn+1];i++)
        R[nn][i]=R[nn+1][i];
    for (i=R[0][nn+1]+1;i<=R[0][nn];i++)
        R[nn][i]=0;
}

void matrixcpyRopR(void)
{
    unsigned int i,b,a,a1;

    a=Rop[0][0];
    free_matrix2(Rop,a+1);
    a=R[0][0];
    for (b=0;b<=a;b++)
    {
        a1=R[0][b];
        Rop[b]=(unsigned int *)alloc_vector(a1+1,sizeof(int));
        for (i=0;i<=a1;i++)
            Rop[b][i]=R[b][i];
    }
}

void matrixcpyRiRop(void)
{
    unsigned int i,b,a,a1;

    a=Ri[0][0];
```

```
        free_matrix2(Ri,a+1);
        a=Rop[0][0];
        for (b=0;b<=a;b++)
        {
            a1=Rop[0][b];
            Ri[b]=(unsigned int *)alloc_vector(a1+1,sizeof(int));
            for (i=0;i<=a1;i++)
                Ri[b][i]=Rop[b][i];
        }
}

void matrixcpyRRi(void)
{
        unsigned int i,b,a;

        if (Ri[0][0]<R[0][0])
        {
            for (b=0;b<=Ri[0][0];b++)
            {
                for (i=0;i<=Ri[0][b];i++)
                    R[b][i]=Ri[b][i];
                for (i=Ri[0][b]+1;i<=R[0][b];i++)
                    R[b][i]=0;
            }
            for (b=Ri[0][0]+1;b<=R[0][0];b++)
                free(R[b]);
        }
        else
        {
            a=R[0][0];
            for (b=0;b<=a;b++)
            {
                for (i=0;i<=Ri[0][b];i++)
                    R[b][i]=Ri[b][i];
                for (i=Ri[0][b]+1;i<=R[0][b];i++)
                    R[b][i]=0;
            }
            for (b=a+1;b<=Ri[0][0];b++)
            {
                R[b]=(unsigned int *)alloc_vector(N+1,sizeof(int));
                initial_vector(R[b],N);
                for (i=0;i<=Ri[0][b];i++)
                    R[b][i]=Ri[b][i];
            }
        }
}

void matrixcpyRRop(void)
{
        unsigned int i,b,a;

        if (Rop[0][0]<R[0][0])
        {
            for (b=0;b<=Rop[0][0];b++)
            {
                for (i=0;i<=Rop[0][b];i++)
                    R[b][i]=Rop[b][i];
                for (i=Rop[0][b]+1;i<=R[0][b];i++)
```

```
                R[b][i]=0;
        }
        for (b=Rop[0][0]+1;b<=R[0][0];b++)
            free(R[b]);
    }
    else
    {
        for (b=0;b<=R[0][0];b++)
        {
            for (i=0;i<=Rop[0][b];i++)
                R[b][i]=Rop[b][i];
            for (i=Rop[0][b]+1;i<=R[0][b];i++)
                R[b][i]=0;
        }
        for (b=R[0][0]+1;b<=Rop[0][0];b++)
        {
            R[b]=(unsigned int *)alloc_vector(N+1,sizeof(int));
            initial_vector(R[b],N);
            for (i=0;i<=Rop[0][b];i++)
                R[b][i]=Rop[b][i];
        }
    }
}

void prepare_liste(void)
{
    unsigned int a;

    liste=(unsigned int *)alloc_vector(N+1,sizeof(int));
    for (a=0;a<=N;a++)
        liste[a]=a;
    liste[0]=N;
}

void calculate_inlen(void)
{
    unsigned int a;
    long cumdem;
    float temp;

    cumdem=0;
    for (a=1;a<=N;a++)
    {
        cumdem+=d[a];
    }
///  fprintf(flog,"\ncumdem=%ld",cumdem);
    temp=cumdem*1.00/cap;
///  fprintf(flog,"\ncumdem/cap=%-8.2f cap=%ld",temp,cap);
    mlow=ceil(temp);
    inlen=sqrt(mlow*N);
///  fprintf(flog,"\ninlen=%u",inlen);
}

void solution_improvement(void)
{
    Qlow=10;
    Qup=20;
```

```
    Q=rand()%(Qup-Qlow+1);
    Q=Qlow+Q;
    Q=N/2;
    alfa=50*N;
    gama=N*mlow/2;
    allocate_op();
    tv=(unsigned int **)alloc_matrix(Q+1,M+P+1,sizeof(int));
    tr=(unsigned int **)alloc_matrix(Q+1,M+P+1,sizeof(int));
    initial_matrix(tr,Q,M+P);
    initial_matrix(tv,Q,M+P);
///  fprintf(flog,"\n**********Solution Improvement**********");
    for (p=1;p<=alfa;p++)
    {
///     fprintf(flog,"\n\niteration=%u",p);
        neighborhood_search();
    }
///  display_route(Rop,Sop,Fop,Fopt);
    free_arrays_op();
}

void allocate_op(void)
{
    Rop[0]=(unsigned int *)alloc_vector(N+1,sizeof(int));
    initial_vector(Rop[0],N);
    Fop=(float *)alloc_vector(N+1,sizeof(float));
    Sop=(long *)alloc_vector(N+1,sizeof(long));
    matrixcpyRopR();
    arraycpy2(Fop,F,N);
    arraycpy3(Sop,slack,N);
    Fopt=Fprev;
}


void il_allocation(void)
{
    RR=(unsigned int ***)d3alloc(inlen+1);
    FF=(float **)d2alloc(inlen+1);
    SS=(long **)d2alloc(inlen+1);
    il=(unsigned int *)alloc_vector(inlen+1,sizeof(int));
    initial_vector(il,inlen);
}

void search_il(void)
{
    unsigned int i,a,found;

    rr=0;
    i=1;
    found=0;
    while ((i<=il[0])&&(found==0))
    {
        a=il[i],
        if (Fprev<FF[a][0])
            found=1;
        else
            i++;
    }
    if ((found==1)||(inlen>il[0]))
```

```
            rr=i;
}

void update_il(void)
{
    unsigned int i,a,k,temp;

    rrr=0;
    if (rr!=0)
    {
        if (il[0]<inlen)
        {
            for (i=il[0];i>=rr;i--)
                il[i+1]=il[i];
            il[0]++;
            il[rr]=il[0];
        }
        else
        {
            rrr=il[inlen];
            for (i=il[0];i>rr;i--)
                il[i]=il[i-1];
            il[rr]=rrr;
        }
    }
}

void allocate_il(void)
{
    unsigned int jj,tt;

    if (rr!=0)
    {
        if (rrr!=0)
        {
            jj=RR[rrr][0][0];
            free_matrix2(RR[rrr],jj+1);
            free(FF[rrr]);
            free(SS[rrr]);
            kk=rrr;
        }
        else
            kk=il[rr];
        jj=R[0][0];
        RR[kk]=(unsigned int **)d2alloc(jj+1);
        FF[kk]=(float *)alloc_vector(jj+1,sizeof(float));
        SS[kk]=(long *)alloc_vector(jj+1,sizeof(long));
        for (tt=0;tt<=R[0][0];tt++)
        {
            jj=R[0][tt];
            RR[kk][tt]=(unsigned int *)alloc_vector(jj+1,sizeof(unsigned int));
        }
    }
}

void assign_il(void)
{
    unsigned int jj,tt;
```

```
        if (rr!=0)
        {
            for (tt=0;tt<=R[0][0];tt++)
            {
                jj=R[0][tt];
                arraycpy(RR[kk][tt],R[tt],jj);
            }
            arraycpy2(FF[kk],F,R[0][0]);
            FF[kk][0]=Fprev;
            arraycpy3(SS[kk],slack,R[0][0]);
        }
}

void keep_solution(void)
{
    search_il();
    update_il();
    allocate_il();
    assign_il();
}

void neighborhood_search2(void)
{
    unsigned int number,b,m,n;

    Fp=MAXFLOAT;
    initialize_neighborhood_search();
    for (it=1;it<=gama;it++)
    {
        choose_routes_vertices();
        if (feasible==1)
        {
            fprintf(flog,"\n\nNeighbor%u:",it);
            fprintf(flog,"\nR1=%u/",R1);
            for(b=1;b<=mu;b++)
                fprintf(flog,"%u,",v1[b]);
            fprintf(flog," mu=%u ",mu);
            fprintf(flog," GF1=%-8.2f",GF1);
            fprintf(flog,"\nR2=%u/",R2);
            for(b=1;b<=pi;b++)
                fprintf(flog,"%u,",v2[b]);
            fprintf(flog," pi=%u",pi);
            fprintf(flog," GF2=%-8.2f",GF2);
            number=R[0][R1]-mu;
            insert(R1,G1,number,v2,pi,&GF1);
            number=R[0][R1]-mu+pi;
            display(R1,G1,number,&GF1);
            if (number>=3)
                twoopt2(G1,number,&GF1);
            number=R[0][R2]-pi;
            insert(R2,G2,number,v1,mu,&GF2);
            number=R[0][R2]-pi+mu;
            display(R2,G2,number,&GF2);
            if (number>=3)
                twoopt2(G2,number,&GF2);
            search_neighborhood();
            free(G1);
```

```
                free(G2);
                free(v1);
                free(v2);
            }

        }
        if (feasible==1)
        {
            if (Fmin==MAXFLOAT)
                Fmin=Ftmin;
            if ((Fmin>Fprev)&&(Fp>Fprev))
            {
                keep_solution();
///             fprintf(flog,"\n====================\n");
            }
            update_route_list();
            check_empty_routes();
            update_tabu_list();
            Fp=Fprev;
            Fprev=Fmin;
            if (Fmin<Fopt)
            {
                Fopt=Fmin;
                matrixcpyRopR();
                arraycpy2(Fop,F,N);
                arraycpy3(Sop,slack,N);
                pop=p;
            }
        }
        else
            fprintf(flog,"\n!!!!!!!!!!!!!!no improvement!!!!!!!!!!!!!");
/*          fprintf(flog,"\nNeighbor Search Results:");
            fprintf(flog,"\nBest neighbor=Neighbor%u",itbest);
            fprintf(flog,"\nTotal Routes:%u",R[0][0]);
            fprintf(flog,"\nRoute Matrix\n");
            for(m=1;m<=R[0][0];m++)
            {
                for(n=0;n<=R[0][m];n++)
                {
                    fprintf(flog,"%u ",R[m][n]);
                }
                fprintf(flog,"0      /");
                fprintf(flog,"slack=%ld  /Froute=%-8.2f\n",slack[m],F[m]);
            }       */
            fprintf(flog,"\n%-6u %-8.2f",p,Fprev);
///         display_tabu();
        fprintf(flog,"%-8.2f %-6u",Fopt,pop);
        free(GG1);
        free(GG2);
        free(t1);
        free(t2);
}

void display_il(void)
{
    unsigned int b,k,l,n,m;

    for (b=1;b<=il[0];b++)
```

```
        {
            m=il[b];
            k=RR[m][0][0];
            fprintf(flog,"\n");
            for (l=1;l<=k;l++)
            {
                fprintf(flog,"\n");
                for (n=1;n<=RR[m][0][l];n++)
                    fprintf(flog,"%u ",RR[m][l][n]);
            }
        }
}

void intensification(void)
{
    unsigned int jj;

    free_arrays();
    Qlow=5;
    Qup=10;
    Q=rand()%(Qup-Qlow+1);
    Q=Qlow+Q;
    Q=N/2;
    allocate_initial();
    for (jj=1;jj<=il[0];jj++)
    {
        fprintf(flog,"\n**********Intensification Solution=%u**********",jj);
        generate_intensification_solution(&jj);
        display_route(R,slack,F,Fprev);
        intensify();
        check_best_initial(&jj);
        free_arrays();
        free_arrays_op();
    }
    free(RR);
    free(SS);
    free(FF);
    allocate_startingpoint();
    prepare_initial();
    fprintf(flog,"\n\n**************BEST SOLUTION***************");
    display_route(R,slack,F,Fprev);
}

void generate_intensification_solution(unsigned int *e)
{
    unsigned int m,n,k,l;


    m=il[*e];
    n=RR[m][0][0];
    for (k=0;k<=n;k++)
    {
        R[k]=(unsigned int *)alloc_vector(N+1,sizeof(int));
        initial_vector(R[k],N);
        l=RR[m][0][k];
        arraycpy(R[k],RR[m][k],l);
    }
    tv=(unsigned int **)alloc_matrix(Q+1,M+P+1,sizeof(int));
```

```
        tr=(unsigned int **)alloc_matrix(Q+1,M+P+1,sizeof(int));
        initial_matrix(tr,Q,M+P);
        initial_matrix(tv,Q,M+P);
        F=(float *)alloc_vector(N+1,sizeof(float));
        slack=(long *)alloc_vector(N+1,sizeof(long));
        for (k=0;k<=N;k++)
            slack[k]=cap;
        arraycpy3(slack,SS[m],n);
        initial_vector2(F,N);
        arraycpy2(F,FF[m],n);
        Fprev=FF[m][0];
        k=RR[m][0][0];
        free_matrix2(RR[m],k+1);
        free(FF[m]);
        free(SS[m]);
///     display_route(R,slack,F,Fprev);
        Rop[0]=(unsigned int *)alloc_vector(N+1,sizeof(int));
        initial_vector(Rop[0],N);
        Fop=(float *)alloc_vector(N+1,sizeof(float));
        Sop=(long *)alloc_vector(N+1,sizeof(long));
        matrixcpyRopR();
        arraycpy2(Fop,F,N);
        arraycpy3(Sop,slack,N);
        Fopt=Fprev;
}

void intensify(void)
{
    alfa=N;
    gama=N*mlow/2;
    for (p=1;p<=alfa;p++)
    {
///     fprintf(flog,"\n\nIteration=%u",p);
        neighborhood_search();
    }
}

void twoopt2(unsigned int *gec, unsigned int elements, float *routcost)
{
    unsigned int a,b,I1,I2,J1,J2,S1,S2,T1,T2,next,last,ahead,index,limit;
    unsigned int *ptr;
    float max1,max;

    ptr=(unsigned int *)alloc_vector(N+1,sizeof(int));
    initial_vector(ptr,N);
    for (b=0;b<elements;b++)
        ptr[gec[b]]=gec[b+1];
    ptr[gec[elements]]=0;
    do
    {
        max=0;
        I1=0;
        for (a=0;a<=elements-2;a++)
        {
            limit=a==0?elements-1:elements;
            I2=ptr[I1];
            J1=ptr[I2];
            for (b=a+2;b<=limit;b++)
```

```
                {
                    J2=ptr[J1];
                    max1=dc[I1][I2]+dc[J1][J2]-(dc[I1][J1]+dc[I2][J2]);
                    if (max1>max)
                    {
                        S1=I1;
                        S2=I2;
                        T1=J1;
                        T2=J2;
                        max=max1;
                    }
                    J1=J2;
                }
                I1=I2;
            }
            if (max>0.0)
            {
                ptr[S1]=T1;
                next=S2;
                last=T2;
                do
                {
                    ahead=ptr[next];
                    ptr[next]=last;
                    last=next;
                    next=ahead;
                }
                while (next!=T2);
                *routcost-=max;
/**             fprintf(flog,"\ntwoopt arcs=(%u,%u)-(%u,%u), ",S1,S2,T1,T2);
                fprintf(flog," cost saving=%-8.2f",max);
                index=0;
                fprintf(flog,"/");
                for (b=0;b<=elements;b++)
                {
                    fprintf(flog,"%u ",index);
                    index=ptr[index];
                }
                fprintf(flog,"0 ");
                fprintf(flog,"/Routcost=%-8.2f",*routcost); **/
            }
        }
        while (max>0);
        index=0;
        for (b=0;b<=elements;b++)
        {
            gec[b]=index;
            index=ptr[index];
        }
        free(ptr);
}
```

REFERENCES

1. Hax, A. C., and, D. Candea, "Hierarchical Production Planning Systems," *Production and Inventory Managemant*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984

2. Scheeweiss, C., "Hierarchical Structures in Organizations: A Conceptual Framework," *European Journal of Operational Research*, Vol. 86, pp. 4-31, 1995.

3. Dempster, M. A. H., M. L. Fisher, L. Jansen, B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnoy Kan, "Analytical Evaluation of Hierarchical Planning Systems," *Operations Research*, Vol. 29, pp. 707-716, 1981.

4. Simichi-Levi, D., "Hierarchical Planning for Probabilistic Distribution Systems in Euclidean Spaces," *Management Science*, Vol. 38, No. 2, pp. 198-211, February 1992.

5. Günther, H. O., "The Design of an Hierarchical Model for Production Planning and Scheduling," Research Paper, Universität Mannheim, pp. 227-260, 1992.

6. Saad G. H., "Hierarchical Production Planning Systems: Extensions and Modifications," *Journal of the Operational Research Society*, Vol. 41, No. 7, pp. 609-624, 1990.

7. Kistner, K. P., and M. Steven, "Applications of Operations Research in Hierarchical Production Planning," in S. Axsäter, C. Schneeweiss and E. A. Silver (Eds.), *Multi-Stage Production Planning and Inventory Control*, pp. 97-113, Springer-Verlag, Berlin, 1986.

8. Graves, S. C., "Using Lagrangean Relaxation Techniques to Sove Hierarchical Production Planning Problems," *Management Science*, Vol. 28, No. 3, pp. 260-275, March 1982.

9. Aardal, K., and T. Larsson, "A Benders Decomposition Based Heuristic for the Hierarchical Production Planning Problem," *European Journal of Operational Research*, Vol. 45, No. 1, pp. 4-14, 1990.

10. Axsäter S., "Aggregation of Product Data for Hierarchical Production Planning," *Operations Research*, Vol. 29, pp. 744-756, 1981.

11. Axsäter S., "On the Feasibility of Aggregate Production Plans," *Operations Research*, Vol. 34, pp. 796-800, 1986.

12. Axsäter S., and H. Jönsson, "Aggregation and Disaggregation in Hierarchical Production Planning," *European Journal of Operational Research*, Vol. 17, pp. 338-350, 1984.

13. Erschler, J., G. Fontan, and C. Merce, "Consistency of the Disaggregation Process in Hierarchical Planning," *Operations Research*, Vol. 34, pp. 464-469, 1986.

14. Tsubone, H., H. Matsuura, and T. Tsutsu, "Hierarchical Production Planning System for a Two-Stage Process," *International Journal of Production Research*, Vol. 29, No. 4, pp. 769-785, 1991.

15. Glover, G., G. Jones, D. Karney, D. Klingman, and J. Mote, "An Integrated Production, Distribution and Inventory Planning System," *Interfaces*, Vol. 9, No. 5, pp. 21-35, 1979.

16. Bloemhof-Ruwaard, J. M., M. Salomon, L. N. Van Wassenhove, "On the Coordination of Product and By-Product Flows in Two-Level Distribution Networks: Model Formulations and Solution Procedures," *European Journal of Operational Research*, Vol. 79, pp. 325-339, 1994.

17. De-Matta, R., "A Lagrangean Decomposition Solution to a Single Line Multiproduct Scheduling Problem," *European Journal of Operational Research*, Vol. 79, No. ?, pp. 25-37, 1994.

18. Goyal, S. K., and S. G. Deshmukh, "Integrated Procurement-Production Systems: A Review," *European Journal of Operational Research*, Vol. 62, pp. 1-10, 1992.

19. Rangan, V. K., and R. Jaikumar, "Integrating Distribution Strategy and Tactics: A Model and an Application," *Management Science*, Vol. 37, No. 11, pp. 1377-1389, November 1991.

20. Pyke, D. F., and M. A. Cohen, "Multiproduct Integrated Production-Distribution Systems," *European Journal of Operational Research*, Vol. 74, pp. 18-49, 1994.

21. Federgruen, A., and Yu-S. Zheng, "Optimal Power-of-Two Replenishment Strategies in Capacitated General Production / Distribution Networks," *Management Science*, Vol. 39, No. 6, pp. 710-727, June 1993.

22. Soumis, F., M. Sauvé, and L. Le Beau, "The Simultaneous Origin-Destination Assignment and Vehicle Routing Problem," *Transportation Science*, Vol. 25, No. 3, pp. 188-200, August 1991.

23. Chien, T. W., A. Balakrishnan, and R. T. Wong, "An Integrated Inventory Allocation and Vehicle Routing System," *Transportation Science*, Vol. 23, No. 3, pp. 67-76, May 1989.

24. Chandra, P., and M. L. Fisher, "Coordination of Production and Distribution Planning," *European Journal of Operational Research*, Vol. 72, pp. 503-517, 1994.

25. Iyogun, P., "Lot-sizing Algorithm for a Coordinated Multi-Item, Multi-Source Distribution Problem," *European Journal of Operational Research*, Vol. 59, No. 3, pp. 393-404, 1992.

26. Iyogun, P., and D. Atkins, "A Lower Bound and an Efficient Heuristic for Multistage Multiproduct Distribution Systems," *Management Science*, Vol. 39, No. 2, pp. 204-217, February 1993.

27. Axsäter S., "Simple Solution Procedures for a Class of Two-Echelon Inventory Problems," *Operations Research*, Vol. 38, No. 1, pp. 64-69, January-February 1990.

28. Laporte, G., "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms," *European Journal of Operational Research*, Vol. 59, pp. 345-358, 1992.

29. Laporte, G., "Selected Readings on Vehicle Routing," Centre de Recherche sur les Transports, Université de Montréal, CRT-96-10, pp. 1-21, February 1996.

30. Laporte, G., Y. Nobert, and M. Desrochers, "Optimal Routing under Capacity and Distance Restrictions," *Operations Research*, Vol. 33, pp. 1050-1073, 1985.

31. Labbé, M., G. Laporte, and H. Mercure, "Capacitated Vehicle Routing Problems on Trees," *Operations Research*, Vol. 39, pp. 616-622, 1991.

32. Laporte, G., H. Mercure, and Y. Nobert, "A Branch-and-Bound Algorithm for a Class of Asymmetrical Vehicle Routing Problems," *Journal of Operational Research Society*, Vol. 43, pp. 469-481, 1992.

33. Fischetti, M., P. Toth, and D. Vigo, "A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem on Directed Graphs," *Operations Research*, Vol. 42, pp. 846-859, 1994.

34. Fischer, M. L., "Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees," *Operations Research*, Vol. 42, pp. 626-642, 1994.

35. Christofides, N., A. Mingozzi, and P. Toth, "State Space Relaxation Procedures for the Computation of Bounds to Routing Problems, " *Networks*, Vol. 11, pp. 145-164, 1981.

36. Balinski, M., and R. Quandt, "On an Integer Program for a delivery Problem," *Operations Research*, Vol. 12, pp. 300-304, 1964.

37. Rao, M. R., and S. Zionts, "Allocation of Transportation Units to alternative Trips- A Column Generation Scheme with out-of-Kilter Subproblems," *Operations Research*, Vol. 16, pp. 52-63, 1968.

38. Foster, B., and D. Ryan, "An Integer Programming Approach to the Vehicle Scheduling Problem," Operational Research Quarterly, Vol. 27, pp. 367-384, 1976.

39. Orloff, C., "Route-Constrained Fleet Scheduling," *Transportation Science*, Vol. 10, pp. 149-168, 1976.

40. Desrosiers, J., F. Soumis, and M. Desrochers, "Routing with Time Windows by Column Generations," *Networks*, Vol. 14, pp. 545-565, 1984.

41. Agarwal, Y., K. Mathur, and H. M. Salkin, "A Set-Partitioning-Based Algorithm for the Vehicle Routing Problem," *Networks*, Vol. 19, pp. 731-750, 1989.

42. Fischer, M. L., and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks*, Vol. 11, pp. 109-124, 1981.

43. Laporte, G., Y. Nobert, and M. Desrochers, "Optimal Routing under Capacity and Distance Restrictions," *Operations Research*, Vol. 33, pp. 1050-1073, 1985.

44. Gillett, B. E., and L. R. Miller, "A Heuristic Algorithm for the Vehicle-Dispatch Problem," *Operations Research*, Vol. 22, pp. 340-349, 1974.

45. Paessens, H., "The Savings Algorithm for the Vehicle Routing Problem," *European Journal of Operational Research*, Vol. 34, pp. 336-344, 1988.

46. Clarke, G., and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Operations Research*, Vol. 12, pp. 568-581, 1964.

47. Christofides, N., and S. Eilon, "An Algorithm for the Vehicle-Dispatching Problem," *Operational Research Quarterly*, Vol. 20, No. 3, pp. 309-318, 1979.

48. Gaudioso, M., and G. Paletta, "A Heuristic for the Periodic Vehicle Routing Problem," *Transportation Science*, Vol. 26, No. 2, pp. 86-92, May 1992.

49. Dell'Amico, M., M. Fischetti and P. Toth, "Heuristics Algorithms for the Multiple Depot Vehicle Scheduling Problem," *Management Science*, Vol. 39, No. 1, pp. 115-125, January 1993.

50. Glover, F., and H. J. Greenberg, "New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence," *European Journal of Operational Research*, Vol. 39, pp. 119-130, 1989.

51. Osman, I. H., "Heuristics for Combinatorial Optimization Problems: Developments and New Directions," *Proceedings of the Second Conference on Information Technology and its Applications, ITA'92*, p.p. 1-25, 1993.

52. Gendreau, M., G. Laporte and J. Potvin, "Metaheuristics for the Vehicle Routing Problem," Centre de Recherche sur les Transports, Université de Montréal, CRT-963, pp. 1-33, January 1994.

53. Alfa, A. S., S. S. Heragu, and M. Chen, "A Three-Opt Based Simulated Annealing algorithm for the Vehicle Routing Problems," *Computers & Industrial Engineering*, Vol. 21, pp. 635-639, 1991.

54. Osman, I. H., "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," *Annals of Operations Research*, Vol. 41, pp. 421-451, 1993.

55. Gendreau, M., A. Hertz and G. Laporte , "A Tabu Search Heuristic for the Vehicle Routing Problem," *Management Science*, Vol. 40, No. 10, pp. 1276-1290, October 1994.

56. Gendreau, M., A. Hertz and G. Laporte , "New Insertion and Post-Optimization Procedures for the Travelling Salesman Problem," *Operations Research*, Vol. 40, pp. 1086-1094, 1992.

57. Taillard, E., "Parallel Iterative Search Methods for Vehicle Routing Problems," *Networks*, Vol. 23, pp. 661-673, 1993.

58. Fisher, M. L., "The Lagrangean Relaxation Method for Solving Integer Proramming Problems," *Management Science*, Vol. 27, No. 1, pp. 1-17, January 1981.

59. Christofides, N., A. Mingozzi, and P. Toth, "The Vehicle Routing Problem," in N. Christofides, A. Mingozzi, P. Toth, and C. Sandi (Eds.), *Combinatorial Optimization*, Wiley, Chichester, pp. 315-318, 1979.

60. Christofides, N., E. Hadjiconstantinou, and A. Mingozzi, A new exact algorithm for the vehicle routing problem based on q-path and k-shortest path relaxations, Working Paper, The Management School, Imperial College, London, 1993.