AN ABSTRACT OF THE THESIS OF

CHIA-HAO CHANG for the DOCTOR OF PHILOSOPHY
(Name) (Degree)

in INDUSTRIAL AND GENERAL ENGINEERING presented on June 7, 1978
(Major Department) (Date)

TITLE: MULTIPERIOD MULTIPLE-ITEM DYNAMIC LOT SIZING PROBLEM
WHEN DISCOUNTS ARE AVAILABLE

Approved: Redacted for privacy

This thesis extends Wagner-Whitin's Planning Horizon Theorem to discount situations in multiperiod multiple-item dynamic lot sizing problems. Three heuristic techniques are developed using the Least Unit Cost Method, Silver-Meal Method, and Inoue-Chang Method. The three techniques are described and compared in terms of their effectiveness in dealing with the dynamic lot sizing problem. These techniques are modified in order to apply to single-item discount situations. The performance of these modified techniques are tested by using Kaimann's data with discount data added and 100 additional sets of randomly generated data. A heuristic program has been developed for each of the three methods. Each program is designed to handle joint-order multiperiod, multiple-item dynamic lot sizing problems. In addition, both no discount and with discount situations are studied in the development of each program. All the above programs were first developed under the assumption that no split orders occurred. A mathematical programming model was then developed for the situations where the split orders were allowed. The difficulties involved in searching solutions using the mixed integer programming model are discussed.

A two-item problem with one discount level is selected to illustrate the developed programs. The performance of the heuristic programs are measured and estimated through the use of dynamic programming techniques applied to some selected special situations as benchmarks. The comparisons of performance of the heuristic programs among themselves are also conducted based upon the costs of reaching solutions and the optimality of the solutions reached by using those programs. In our testing examples, the average costs of solutions reached by the heuristic methods based upon the Least Unit Cost Method, Silver-Meal Method, and Inoue-Change Method are -$560.9, -$1475.28, and -$1742.36 respectively. The average CPU times for each heuristic program to reach a solution for a 12-period two-item single discount problem are 0.052 sec., 0.064 sec., and 0.054 sec. respectively. A conclusion is reached that the heuristic program based upon the Inoue-Chang Method has significant advantages over other programs.

MULTIPERIOD MULTIPLE-ITEM DYNAMIC LOT SIZING PROBLEM WHEN DISCOUNTS ARE AVAILABLE

by

CHIA-HAO CHANG

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

DOCTOR OF PHILOSOPHY

Completed June 1978

Commencement June 1979

APPROVED:

Redacted for privacy

Professor of Industrial and General Engineering
in charge of major

Redacted for privacy

Head of Department of Industrial and General
Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented June 7, 1978

Thesis typed by CAMPUS PRINTING & COPY CENTER for

CHIA-HAO CHANG

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to Professor Inoue. During the last two years he had been terribly busy, yet he arranged every possible time to advise my study. I count myself most fortune to have him as my advisor.

I like also to thank Dr. Patterson, Dr. McDowell, Dr. Goheen, Dr. McFarlane, Dr. Oberhettinger, and Dr. Smith. They have spent their invaluable time on my program. Specially I would like to express my sincere gratefulness to Dr. Goheen and Miss Nora Cheung. They gave me their advices on my writing.

I particularly feel indebted to Jane, my wife. I decided to stay at Corvallis this last year to finish my degree, and left her at Minneapolis. Although we can not see each other very frequently, she never forget to send me her encouragement constantly. Besides, she delivered our first child, Ming-chee, ten days before my oral final examination. During the preparation of this thesis, many time I had really felt very frustrated, it was her love that supported me to continue my work. Here I like to dedicate this thesis, a symbol of an important phase of my education, to her with love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

MULTIPERIOD MULTIPLE-ITEM DYNAMIC LOT SIZING PROBLEM WHEN DISCOUNTS ARE AVAILABLE

CHAPTER I

INTRODUCTION

Lot Sizing Problem and EOQ

In practical application of production and inventory control methodology, a decision maker will often encounter questions about when and in what quantity he should manufacture or purchase certain products to satisfy the demands. The first question can be answered with certainty provided that the required demands and their corresponding lead times are known. The second question may be solved through lot sizing technique which figures the order size based on the future demand's magnitude and timing.

With the assumption that demand patterns are uniform and stocks are gradually depleted, Wilson's traditional Economic Order Quantity is commonly used to find the order quantity. This approach works fairly well in static cases.

Since Wilson's Economic Order Quantity is built upon the assumption of uniform demand pattern and figures the order size based on the average demand without considering the "timing" of the demand, the outcome tends to be unsatisfactory, not economical, may even be disastrous when the basic assumptions are unrealistic. For example, in a manufacturing environment, the demand pattern of the components of assembled products is typically not uniform, and depletion is not gradual. The Economic Order Quantity turns out to be a poor ordering quantity when it faces such discrete lumpy demand patterns.

Dynamic Lot Sizing Problem

Because of the failure of Economic Order Quantity to deal with the frequently encountered discrete lumpy demands, the interest in recent years has gradually shifted to discrete lot sizing techniques which make no assumption of uniform demand patterns. Several such techniques are listed in Table 1-1.

Lot by Lot
Period Order Quantity
Least Total Cost
Part Period Balancing
Least Unit Cost
Silver-Meal Method
Inoue-Chang Phase-1 Method
Inoue-Chang Phase-1 and Phase-2 Method
Wagner and Whitin's Algorithm

Table 1-1: Discrete Lot Sizing Techniques

In all cases, the planning horizon is divided into periods, which are often counted in units of weeks. The demand rate in each period is assumed to be deterministic. When the demands vary over time, the associated lot sizing problem is called a dynamic lot sizing problem. The objective of a dynamic lot sizing technique is to determine the proper lot size to fill the non-uniform demand requirements, and to decide how many of periods of requirements should be combined into a single lot. Backlog is generally not allowed, especially in an assembly process. A delay to supply a part at the required time will cause delay of the whole process. Such costs are often very high.

Among all the dynamic lot sizing techniques, the Wagner-Whitin's algorithm (Wagner and Whitin, 1958) is the only one that guarantees a minimum total cost inventory management scheme which satisfies dynamic demand patterns. The algorithm uses a dynamic programming method to compute and compare all possible combinations of solutions. Though the algorithm requires large computational efforts, its guaranteed optimality makes it a valuable benchmark against which the performance of other techniques is measured (Kaimann, 1969; Berry, 1972; Silver and Meal, 1973; Ruch, 1976; Chang and Inoue, 1977).

Discount Consideration

Although many scholarly efforts have been expended in the general area of lot sizing research, one related area has received relatively little attention. This is the study of the optimum purchase quantity and timing decision when discounts are available.

The supplier may offer different levels of discounts when larger quantities are purchased. The price differentials may be substantial. If a solution indicated by a lot sizing technique comes close to a discount level break, it will be easy to adjust the lot sizing quantity in order to take advantage of its discount saving. Some solutions may not be that obvious, so the potential discount saving must be balanced against the extra holding cost of carrying more inventory over a longer period and determining what is the correct quantity to purchase.

In fact, a study of purchasing quantity and timing when discounts are available often provides significant cost reduction. It has been pointed out that it is not uncommon to find unit price reduction in excess of 50% for agreements to purchase in increased quantities (Whybark, 1977). Unfortunately, almost all discrete dynamic lot sizing techniques assume no discounts. Callarman and Whybark did research on the comparison of several dynamic lot sizing techniques on the single-item demand patterns (Callarman and Whybark, 1977). They allowed some techniques to order with split lot while forbidding the Wagner-Whitin's algorithm to do the same, and found that some heuristic techniques performed superior to the Wagner-Whitin's dynamic programming approach (Callarman and Whybark, 1977).

Multiple-Item Consideration

Most of the presently known dynamic lot sizing techniques deal with single-item problems. They can be used to deal with some multiple-item problems if either one of the following conditions is fulfilled:

1. There are multiple items, but the production or inventory processes, resources, and capacities involved are such that each item can be planned independently.
2. There are multiple items, but the decision is based only on the aggregated level without specifying production or

inventory levels for individual items.

But those single-item lot sizing techniques cannot solve all the problems. There are a lot of situations where many items are involved. They may either use common facilities, labor or material as in many production problems, or have a joint set-up cost as in the lot sizing problem. Those items must be considered jointly instead of being planned independently. In addition, since the decision will depend on the solutions for the individual items, the problem cannot be solved through aggregated planning techniques. These kinds of multiple-item problems are usually characterized by considerable computational difficulty (Johnson and Montgomery, 1974). A number of authors have worked with multiple-item problems (Shu, 1971; Nocturne, 1973; Chern, 1974; Andres and Emmons, 1975; Silver, 1975; Zoller, 1977). They either assume that the demand rates of items are constant or the demand rates follow some generalized mathematical functions. Therefore, in past studies, the decisions assumed continuous review rather than periodical review. In our study, as in the dynamic lot sizing problem, the generalized mathematical functions to describe the demands are not considered to be known. This necessitates a periodic review approach. Eisenhut presented a heuristic algorithm to deal with the multiple-item dynamic lot sizing problem with capacity constraints, but he did not consider the case when discounts are available (Eisenhut, 1975). No literature is found to work with the joint order multiple-item dynamic lot sizing problem when the discounts are available, and that is the topic of this thesis.

Engineering Relevance

The problem presented here is a deterministic multiperiod multiple-item joint order cost problem, and discounts are assumed available. The assumptions made are as follows:

1. There are K items involved.
2. Demand for ith item at jth period, D_{ji}, is deterministic.
3. Demand is dynamic, or said to vary with time.
4. The ordering cost of all items is jointed.
5. Holding costs are at a constant rate h_i for the ith item per unit per period.
6. The backlog is not allowed.
7. Lead time is negligible.
8. Order is under periodic review.
9. There is no initial stock.
10. The planning horizon is finite.
11. There is no split lot allowed.

In real world problems, any item always has a lead time, and the initial stock is unlikely to be zero. But only through moving the item demands ahead of lead time periods, can the lead time of newly generated demand rates be treated as zero, and by subtracting demands from the on hand initial stock, make the results as if there were no initial stock. When a family of items are produced together through a common set-up procedure, or several items have close lead times, these items may be ordered simultaneously through a joint order in order to take advantage of the economy that can result from such actions. A common set-up procedure for a family item can reduce the set-up time and increase the productivity. Similarly, a joint order can reduce the ordering time and save the ordering cost.

Outlines of this Thesis

The planning horizon theorem was adopted to reduce computational difficulty in the search for an optimum solution of a dynamic lot-sizing problem through the dynamic programming approach. The theorem is described in Chapter II and extended to the cases where the discounts are available. The necessary condition to make the theorem valid is also presented.

The third chapter deals with single-item dynamic lot-sizing techniques. Traditionally used techniques are presented. The author's newly developed technique is also introduced. Those techniques are compared by using a set of well-known standard data, as well as 100 sets of computer generated random data. The modification of some techniques to the discount situation is also discussed and developed. Again, some testing data are generated, and the techniques are tested and compared.

The joint order multiperiod multiple-item dynamic lot sizing problems when discounts are available are discussed in the fourth chapter. Three heuristic programs are developed based on different criteria. The flowchart and details of the approaches are presented in the same chapter. Also, an optimization algorithm is developed using the dynamic programming method. The extension of planning horizon theorem in a multiple-item discount situation is discussed in order to reduce the computational effort in searching the optimum solution. The fifth chapter discusses the same situation, but the assumption of not allowing split order is eliminated. A network to represent a generalized model is illustrated. Mathematical programming is used to model the problem. The approach to solve such a model is also discussed in the fifth chapter. In the sixth chapter, numerical examples are presented. Two-item problem with one discount level for each is selected as the example to illustrate the developed programs. The testing data are generated and the performance of different heuristic programs are estimated by using the dynamic programming techniques in some selected situations as benchmarks. The results are compared and evaluated. The conclusions are presented in the last chapter. The potential areas for further study and investigation are also recommended.

CHAPTER II

PLANNING HORIZON THEOREM

Wagner-Whitin's Planning Horizon Theorem

Of all presently known dynamic ordering rules, Wagner-Whitin's dynamic programming approach (Wagner and Whitin, 1958) is the only method that will guarantee an optimum solution. This method searches all possible combinations of ordering quantities at different periods and finds the best combination. The process requires a large amount of computation.

To simplify the searching process, Wagner-Whitin's program uses the Planning Horizon Theorem to eliminate combinations that need not be considered during the searching process. The theorem applies to a situation where a decision must be made between ordering the quantity P_{t_n} at the $t_n{}^{th}$ period versus ordering it at time t_k. The theorem can be stated officially as follows:

> Planning Horizon Theorem: If in the forward algorithm the minimum cost decision at t_n occurs for $P_{t_k} > 0$, $t_k < t_n$, then in periods $t > t_n$ it is sufficient to consider only periods j so that $t_k \leq j \leq t$. (Riggs & Inoue, p. 317, 1975)

It can be written in a recursive form as: Define V_j^* as the minimum cost of ordering and holding the demands up to j^{th} period. The recursive function will then be:

$$V_j^* = \min_{i \leq j} (L_{ij} + V_{i-1}^*), \quad V_{-1}^* = 0$$

$$L_{ij} = H \sum_{l=i}^{j} D_l \cdot (l - i) + O \cdot \delta_{ij}$$

H: Holding cost
O: Ordering cost

$$\delta_{ij} = \begin{cases} 0 \text{ if } \sum_{l=i}^{j} D_l = 0 \\ 1 \text{ if } \sum_{l=i}^{j} D_l > 0 \end{cases}$$

Planning Horizon Theorem can then be redefined as: If in the forward algorithm, the $V_n^* = L_{kn} + V_{k-1}^*$ for ordering and holding the demands up to t_nth period, then in periods $t > t_n$,

$$V_t = \min_{k \leq i \leq t} (L_{it} + V_{i-1}^*), \quad V_{-1}^* = 0$$

Numerical Example of No Discount Situation

Let's use the following example to illustrate the Planning Horizon Theorem. Suppose the demands for the next six periods are the following:

Periods	1	2	3	4	5	6
Demands	100	160	40	200	120	30

The ordering cost is $100.00, and the holding cost is $2.00 per period per unit. The Figure 2-1 below shows the computational process to search for the optimum solution. The * marks the optimum ordering of demands from the first period up to that period. The X represents the terms that need not be considered because of the Planning Horizon Theorem. The optimum solution will then be:

Placing Order at Periods	1	2	4	5
Ordering Quantities	100	200	200	150

Total Cost will be $540.00.

Planning Period	Demands	Placing Order At					
		Period-1	Period-2	Period-3	Period-4	Period-5	Period-6
1	100	(100) $ 100*					
2	160	(260) $ 420	(160) $ 200*				
3	40	X	(200) $ 280*	(40) $ 300			
4	200	X	(400) $1080	(240) $ 700	(200) $ 380*		
5	120	X	X	X	(320) $ 620	(120) $ 480*	
6	30	X	X	X	X	(150) $ 540*	(30) $ 580
Order Shall Be Placed At:		✓	✓		✓	✓	
Ordering Quantities:		100	200		200	150	

Figure 2-1 An Example of Planning Horizon Theorem

Quantity Discount Situation

When discounts are available, the situation will be changed. The total cost is then the sum of ordering costs, holding costs, and the items' price subtracting the saving from the discounts. Since the items' price can be taken as a fixed value, which will not affect the planning decision, the objective of the planning decision will then be to search the optimum solution with the minimum sum of the ordering costs, holding costs, subtracting the discount saving. Because the problem now involves the discount saving, and this varies with the ordering quantities, those combinations that earlier needed not be considered from the conclusion of Planning Horizon Theorem (at no discount situation), can no more be neglected.

A Quantity Discounted Example

Let's use the previous numerical example with the addition of discounts:

Assume:	Ordering Quantity	Discount
	0 - 99	$0/unit
	100 - 299	$2/unit
	300 - 499	$4/unit
	500 -	$6/unit

From the following Figure 2-2, the optimum solutions will be either to order 100 units of items at period-1, and 550 items at period-2, or

Planning Period	Demands	Placing Order at					
		Period-1	Period-2	Period-3	Period-4	Period-5	Period-6
1	100	(100) $-100*					
2	160	(260) $ 100	(160) $ 0*				
3	40	(300) $-620*	(200) $-320	(40) $ 100			
4	200	(500) $-1220*	(400) $-720	(240) $ 20	(200) $-920		
5	120	(620) $-980	(520) $-1520	(360) $-460	(320) $-1560*	(120) $-1360	
6	30	(650) $-860	(550) $-1460*	(390) $-400	(350) $-1440	(150) $-1360	(30) $-1460*

Figure 2-2 An Example of Planning Horizon Theorem With Discounts

to order 300 units of items at period-1, 320 units of items at period-4 and 30 units of items at period-6. Both ways reach the same minimum cost.

This example demonstrates that the traditional Planning Horizon Theorem cannot be applied to the situations where the discounts are available. For example, the optimum Planning for the first two periods is to place an order of 100 units at the 1st period, and 160 units at the 2nd period. If we were to follow the Planning Horizon Theorem, the conclusion would have been that for the further planning we would not need to consider ordering any other amount of demands at the first period. Our example, on the other hand, showed that when the planning period extended to the third period, the optimum solution specified ordering 300 units at the first period. This is indicative of the significant saving from the discounts.

Planning Horizon Theorem Applied to Situations with Discounts

In general, as we see, the implication of traditional Planning Horizon Theorem cannot be applied to the situations when discounts are available. However, there are cases when the implication of traditional Planning Horizon Theorem still can remain valid.

Assume, for example, a situation of planning for n periods with demands $D_1, D_2, \ldots, D_n$. There are J levels of discounts, $G_1, G_2, \ldots, G_J$, available for ordering quantities greater than or equal to $B_1, B_2, \ldots, B_J$. Both series are monotonically nondecreasing.

Suppose at the optimum situation it will order $\sum_{l=k}^{n} D_l$ at period

t_k for planning up to period t_n. In order to extend the planning horizon up to $t_m>t_n$, a decision must be made between ordering the quantity $\sum_{l=i}^{m} D_l$ at t_i versus ordering $\sum_{l=k}^{m} D_l$ at period t_k to reach an optimum solution, i, $k \leq m$.

Theorem 2-1: If in the forward algorithm the minimum cost decision of planning up to period t_n is through ordering quantity $P_{t_k} = \sum_{l=k}^{n} D_l$, $t_k \leq t_n$, and $P_{t_k} \geq B_J$, then in order to extend the planning horizon to the period $t_m>t_n$, it is sufficient to consider only period j, $t_k \leq j \leq t_m$.

Proof: Let C_w represent the total cost (ordering cost plus holding cost subtracting discount reduction in order to fulfill the demands) of planning up to the period t_n through ordering quantity $P_{t_w} = \sum_{l=w}^{n} D_l$ at the period t_w, $C_k \leq C_i$ for $i \neq k$.

When the planning horizon extends to the period $t_m>t_n$, the cost through ordering $P_{t_k} = \sum_{l=k}^{n} D_l$ at t_k is:

$$C'_k = C_k + \sum_{l=n+1}^{m} D_l \, H (l - k) + [G(\sum_{l=k}^{n} D_l) \sum_{l=k}^{n} D_l - G(\sum_{l=k}^{m} D_l) \sum_{l=k}^{m} D_l]$$

where G(Q) represents the discount rate for ordering quantity Q at one time.

For the period $j<t_k$, the cost will be:

$$C'_j = C_j + \sum_{l=n+1}^{m} D_l \, H (l - j) + [G(\sum_{l=j}^{n} D_l) \sum_{l=j}^{n} D_l - G(\sum_{l=j}^{m} D_l) \sum_{l=j}^{m} D_l]$$

Since $\sum_{l=k}^{n} D_l \geq B_J$, and $\sum_{l=j}^{m} D_l$, $\sum_{l=j}^{n} D_l$, $\sum_{l=k}^{m} D_l$ all are greater than $\sum_{l=k}^{n} D_l$, which implies

$$G\Big(\sum_{l=k}^{n} D_l\Big) = G\Big(\sum_{l=j}^{n} D_l\Big) = G\Big(\sum_{l=k}^{m} D_l\Big) = G\Big(\sum_{l=j}^{m} D_l\Big) = G_J$$

and

$$C'_j - C'_k = (C_j - C_k) + \sum_{l=n+1}^{m} D_l \, H (k - j) + G_J \sum_{l=n+1}^{m} D_l - G_J \sum_{l=n+1}^{m} D_l$$

$$= (C_j - C_k) + \sum_{l=n+1}^{m} D_l \, H (k - j)$$

Because $C_j \geqslant C_k$, and $k>j$, we can therefore draw the conclusion that C'_j is always greater than C'_k. Thus, to plan the period $t>t_n$, it is sufficient to consider only periods j where $t_k \leqslant j \leqslant t$.

Theorem 2-2: Assume in the forward algorithm, the minimum cost decision is through ordering $\sum_{l=k}^{n} D_l>0$ at the period t_k, $t_k \leqslant t_n$. The discount levels are based on the ordering quantities $B_1, B_2, \ldots, B_J$. If $\sum_{l=k}^{n} D_l<B_J$, it is possible to find a set of discount rates $G(B_i)$, $i = 1, 2, \ldots, J$, such that when the planning horizon extends to the period $t_m>t_n$, C'_j will be less than C'_k, where $j<t_k<t_m$.

Proof: From the Proof of Theorem 2-1 we have:

$$C'_j - C'_k = C_j - C_k + \sum_{l=n+1}^{m} D_l \, H (k - j) + \Big[G\Big(\sum_{l=j}^{n} D_l\Big) \sum_{l=j}^{n} D_l$$

$$- G\Big(\sum_{l=j}^{m} D_l\Big) \sum_{l=j}^{m} D_l \Big] - \Big[G\Big(\sum_{l=k}^{n} D_l\Big) \sum_{l=k}^{n} D_l - G\Big(\sum_{l=k}^{m} D_l\Big) \sum_{l=k}^{m} D_l\Big]$$

Since $\sum_{l=k}^{n} D_l<B_J$, let's set the discount rates as the following:

$$G\Big(\sum_{l=k}^{n} D_l\Big) = G\Big(\sum_{l=k}^{m} D_l\Big) = G(B_f)$$

$$G\left(\sum_{l=j}^{m} D_l\right) = G\left(\sum_{l=j}^{n} D_l\right) + \delta = G(B_g) \qquad f<g\leq J$$

and let

$$x_1 = C_j + C_k, \quad x_2 = \sum_{l=n+1}^{m} D_l \, H(k-j)$$

we will have

$$C'_j - C'_k = x_1 + x_2 + \Big[G\left(\sum_{l=j}^{n} D_l\right) \sum_{l=j}^{n} D_l - G\left(\sum_{l=j}^{n} D_l\right) \sum_{l=j}^{n} D_l - \delta \sum_{l=j}^{m} D_l$$

$$- G\left(\sum_{l=j}^{n} D_l\right) \sum_{l=n+1}^{m} D_l + G\left(\sum_{l=k}^{n} D_l\right) \sum_{l=n+1}^{m} D_l \Big]$$

because $G(Q)$ is a monotonically nondecreasing series,

$$C'_j - C'_k = x_1 + x_2 - \delta \sum_{l=j}^{m} D_l$$

The δ can be chosen as

$$\delta = (x_1 + x_2 + \varepsilon) / \sum_{l=j}^{m} D_l \qquad \varepsilon>0$$

which concludes

$$C'_j < C'_k \, .$$

For a situation to plan for N periods of demands $D_1, D_2, \ldots, D_N$, if in the forward algorithm the minimum cost decision of planning up to period t_n is through ordering quantity $P_{t_k} = \sum_{l=k}^{n} D_l$ at the period t_k,

$t_k \leq t_n$, and if the discount levels are based on the quantity B_1, B_2, . . ., B_J, the Theorem 2-1 represents the sufficient condition in order to get a minimum cost decision through considering only period j when the planning horizon extends to $t_m > t_n$, $t_k \leq j \leq t_m$, and the Theorem 2-2 represents a necessary condition.

It should notice that theoretically the Theorem 2-2 works on the hypothetical cases, but in the real life cases the factor δ will be limited by the item's original price, otherwise we may have the discount rate that gives a negative price, which generally will not happen.

Generally speaking, the Planning Horizon Theorem is not appropriate for a case when the discounts are available. However, if certain conditions are fulfilled the implication of the Planning Horizon Theorem still can apply to the case, and that will save the computational time and the storage area in searching the optimum solution. The numerical example using the Planning Horizon Theorem applied to the discount situation will be discussed at next chapter.

CHAPTER III

SINGLE-ITEM DYNAMIC LOT SIZING TECHNIQUES

Single-Item Dynamic Lot Sizing Problem

Wilson's Economic Order Quantity has commonly been used to find the order quantity, and this approach works fairly well in static cases. However, in the manufacturing environment, and many other real life environments as well, the demand patterns are considered discrete and changing with time. The EOQ formula is based upon the assumption that the demand pattern is uniform; when faced with dynamic lot sizing problems, the use of the EOQ often leads to unsatisfactory solutions. There are many techniques developed to cope with such dynamic demand patterns, including:

1. Fixed Order Quantity, variable order interval
2. Fixed Period Requirement, variable order quantity
3. Lot-for-Lot
4. Economic Order Quantity (EOQ) Formula
5. Period Order Quantity
6. Ruch's Method
7. Least Unit Cost (LUC) Method
8. Silver-Meal Method
9. Eisenhut's Method
10. Least Total Cost (LTC) Method
11. Part-Period Balancing (PPB) Method

12. Inoue-Chang Phase-1 Method
13. Inoue-Chang Phase-1 and Phase-2 Method
14. Wagner-Whitin Method.

Classification of the Techniques

Among techniques using these methods, those using methods 7 to 14 allow both the lot size and the interval change for each order to be placed. And these eight techniques can be classified into four kinds of approaches. From 7 to 13, these are heuristic techniques following three kinds of approaches to the dynamic lot sizing problem. Lease Unit Cost finds the solutions based on the local minimal unit cost. Silver-Meal Method and Eisenhut Method are based on the local minimal total cost per period. Technique based on methods 10 to 13 are based on the assumption that the optimal solution locates when the ordering cost is close to the holding cost, and add some modifications and improvements. The Wagner-Whitin method represents the fourth kind of approach; it uses dynamic programming approach to search all the possible ways to meet the demands and determining the optimal solution as the one with minimum cost. Since the Wagner-Whitin method guarantees an optimum solution, it is often used as a benchmark to measure the performance of other techniques.

Among those heuristic techniques, those based upon Least Unit Cost, Silver-Meal Method, and Inoue-Chang Method are selected to represent three kinds of different approaches, and the techniques are developed in extension to the situation when discounts are available and when multiple-items are involved.

Comparisons of Dynamic Lot Sizing Techniques Using Kaimann's Data

Along with the development of different dynamic lot sizing techniques, a number of papers have been presented on the analysis and comparison of techniques. At the early stage, the emphasis was on the comparison of the dynamic programming approach with the traditional EOQ formula (Kaimann, 1969; Gorenstein, 1970; Gleason, 1971). As more and more heuristic dynamic lot sizing techniques were developed, interest was shifted to the comparison of heurisitc approaches using the dynamic programming model as the benchmark (Silver and Meal, 1973; Orlicky, 1975; Ruch, 1976; Chang and Inoue, 1977). A set of standard data (Table 3-1), developed by Kaimann, has been widely used as the demand patterns to test the different techniques. The data are varied along two dimensions: the coefficient of variation of the demand patterns, and the ratio of the economic order quantity to the average period demand (Table 3-2). The first parameter describes the degree of variation in the demand data in terms of the ratio of the standard deviation of weekly demand to the average weekly demand. The more the demand pattern tends to be uniform, the smaller the value of the parameter will be; the more the demand pattern tends to be "lumpy" (Berry, 1972), the larger the value of the parameter will be. The second parameter measures the degree of mismatch between integral multiples of product demand, and is used to measure the "spikeness" in the demand (Berry, 1972). The results of the comparison among the techniques are displayed both in terms of total

Week	1	2	3	4	5
1	92	80	50	10	0
2	92	100	80	10	0
3	92	125	180	15	0
4	92	100	80	20	0
5	92	50	0	70	0
6	92	50	0	180	1105
7	92	100	180	250	0
8	92	125	150	270	0
9	92	125	10	230	0
10	92	100	100	40	0
11	92	50	180	0	0
12	93	100	95	10	0
Coefficient of Variation:	0	.293	.718	1.41	3.31

Table 3-1. Demand Patterns

$EOQ/\bar{D}$	EOQ	Ordering Cost	Holding Cost Per Unit Per Week
.73	67	$ 48	$2
1.00	92	92	2
1.14	105	120	2
1.50	138	206	2
1.82	166	300	2

Table 3-2. Inventory Model Parameters

SOURCE: Kaimann, R. A., "EOQ vs. Dynamic Programming - Which One to Use for Inventory Control?", Production and Inventory Management, 4th Qtr., 1969.

inventory cost performance and the percentage increases over Wagner-Whitin's solution (Table 3-3, 3-4). The comparison shows that among all heuristic methods, the two-phase Inoue-Chang method results in having as low cost as any other approach including Wagner-Whitin. Even when only the first phase is used, the results are generally superior to all other heuristic methods. For details, the reader is referred to Table 3-3, 3-4.

Comparisons Using Randomly Generated Data

One weak point of the above comparison is the number of Kaimann's data. There are only five sets of data. In order to get a clearer picture of the comparisons of different methods, 100 sets of demand patterns are generated in coping with Kaimann's data. Each set of data contains 12 demands with the sum to be 1105, which will lead to the same EOQ because EOQ depends only on the average demand. The 100 sets of demands are randomly generated according to the distribution of zero demands. The distribution functions used have the following characteristics:

Data Set Distribution	% of Zero-Demand
20%	0
20%	10
20%	20
20%	30
20%	40

A list of data is given in Appendix A. The coefficients of variations

EOQ/$\overline{D}$ Ratio	Procedure	Coefficient of Variation 0	.293	.718	1.41	3.31
.73	EOQ	1681	1681	1585	1633	1153
	LUC	1681	1681	1737	1597	1153
	S and M	1681	1681	1557	1597	1153
	W-W Alg.	1681	1681	1557	1589	1153
	I and C					
	Phase I	1681	1681	1557	1589	1153
	Phase I & II	1681	1681	1557	1589	1153
1.0	EOQ	2209	2915	2601	2655	1197
	LUC	2209	2209	2133	2061	1197
	S and M	2209	2209	1953	1981	1197
	W-W Alg.	2209	2209	1953	1941	1197
	I and C					
	Phase I	2209	2209	1953	1961	1197
	Phase I & II	2209	2209	1953	1941	1197
1.14	EOQ	3612	3085	3275	3105	1225
	LUC	2545	2605	2425	2285	1225
	S and M	2545	2545	2205	2165	1225
	W-W Alg.	2545	2505	2205	2145	1225
	I and C					
	Phase I	2545	2505	2205	2165	1225
	Phase I & II	2545	2505	2205	2145	1225
1.5	EOQ	3859	4873	3747	3799	1311
	LUC	3447	3353	3113	2941	1311
	S and M	3447	3541	2871	2701	1311
	W-W Alg.	3447	3353	2871	2681	1311
	I and C					
	Phase I	3447	3353	2871	2681	1311
	Phase I & II	3447	3353	2871	2681	1311
1.82	EOQ	5119	5435	4927	4653	1405
	LUC	4011	4155	3745	3705	1405
	S and M	4011	4055	3455	3245	1405
	W-W Alg.	4011	4055	3435	3245	1405
	I and C					
	Phase I	4011	4055	3435	3245	1405
	Phase I & II	4011	4055	3435	3245	1405

Table 3-3. Comparison Table of Inventory Cost Performance

SOURCE: Berry, W. L., "Lot Sizing Procedures for Requirements Planning Systems: A Framework for Analysis", Production and Inventory Management, 2nd Qtr., 1972.

EOQ/$\bar{D}$ Ratio	Procedure	0	.293	.718	1.41	3.31
.73	EOQ	0	0	2.05	2.76	0
	LUC	0	0	11.56	0.50	0
	S and M	0	0	0	0.50	0
	I and C					
	Phase I	0	0	0	0	0
	Phase I & II	0	0	0	0	0
1.0	EOQ	0	31.96	33.17	36.78	0
	LUC	0	0	9.21	6.18	0
	S and M	0	0	0	2.06	0
	I and C					
	Phase I	0	0	0	1.03	0
	Phase I & II	0	0	0	0	0
1.14	EOQ	41.92	23.15	48.52	44.75	0
	LUC	0	3.99	9.97	6.53	0
	S and M	0	0	0	0.93	0
	I and C					
	Phase I	0	0	0	0.93	0
	Phase I & II	0	0	0	0	0
1.50	EOQ	11.95	45.33	30.51	41.70	0
	LUC	0	0	8.43	9.70	0
	S and M	0	5.60	0	0.74	0
	I and C					
	Phase I	0	0	0	0	0
	Phase I & II	0	0	0	0	0
1.82	EOQ	27.64	34.03	44.13	49.42	0
	LUC	0	2.46	9.02	14.17	0
	S and M	0	0	0.58	0	0
	I and C					
	Phase I	0	0	0	0	0
	Phase I & II	0	0	0	0	0

Table 3-4. Percentage Increase Comparison Table

SOURCE: Berry, W. L., "Lot Sizing Procedures for Requirements Planning Systems: A Framework for Analysis", Production and Inventory Management, 2nd Qtr., 1972.

are ranged from .324 to 1.601 with an average .843

The comparison (Table 3-5, 3-6) shows that the Inoue-Chang method is again significantly superior to the Silver-Meal, Least-Unit, and EOQ methods. Both Inoue-Chang and Silver-Meal methods show a tendency to get nearer to the optimum solution as the ratio of ordering cost to the holding cost becomes smaller. But such tendency does not happen to the Least Unit Cost method, where the number of the optimal results raises from 7% when the ordering cost is $300.00 up to 64% when the ordering cost is $48.00, and the improvement in the average cost over the optimal results is not very steady and significant. The EOQ method, as expected, leads to non-optimal solutions for all the testing data. That is because the assumptions of the method based on the EOQ will not be valid for the dynamic demands. The performance of total cost over the optimal solution is increasing along with the decreasing ratio of ordering cost to holding cost. That behavior is just opposite to other methods. The traditional EOQ method has the worst results compared with the other dynamic lot sizing methods for the planning of dynamic demands, and such result is also expected.

Modification of Single-Item Dynamic Lot Sizing Techniques when Discounts are Available

When the discounts are offered, the costs that go into the price of the item vary extensively throughout the quantity ranges. In order to get a better solution, such potential advantages from the discounts must be put into consideration in order to get the "right quantity."

Ordering Cost/ Holding Cost	PH-1	PH-2	SM	LUC	EOQ
$300/$2	81%	91%	57%	7%	0%
$206/$2	87%	97%	68%	8%	0%
$120/$2	96%	99%	89%	17%	0%
$ 92/$2	100%	100%	92%	21%	0%
$ 48/$2	100%	100%	100%	64%	0%

Table 3-5. The Frequency of Optimum Results

	Average % of Cost Over the Optimum Results				
Ordering Cost/ Holding Cost	PH-1	PH-2	SM	LUC	EOQ
$300/$2	0.601	0.204	1.486	19.919	68.557
$206/$2	0.335	0.060	1.047	22.558	100.512
$120/$2	0.093	0.002	0.273	26.026	112.930
$ 92/$2	0	0	0.120	25.585	139.004
$ 48/$2	0	0	0	16.769	171.549

Table 3-6. The Comparison of Average Percentage Costs Over The Optimum Solutions

Generally speaking, there are two kinds of discounts widely used in industrial and business environments. One is called "All Unit Discounts." Under such discounts, the reduced price is applied to all units when the quantity of the order exceeds some discount level. Another kind of discount is called "Incremental Discount." Under this kind of discount offering, the price reduction only applies to the units above the last discount level when the current order exceeds the next discount level. "It is considered that the case of 'All Unit Discounts' is considerably more difficult to solve, even though it is the form most often found in the industry." (Whybark, 1977) The discount we are going to deal with is the first kind, the All Unit Discount.

With the availability of discounts, the algorithms of dynamic lot sizing techniques will have to consider the trade-off between the potential of reducing the purchasing cost, and increasing the holding cost for the increased inventory.

Let's keep the assumptions of the problem unchanged. A further assumption being added to the problem, namely: The item has J levels of discounts available, B_i, i=1, 2, . . ., J, with the cost reduction rates, G_i, i=1, 2, . . ., J. In all the procedures, the algorithms will first find each order quantity assuming no discounts available. Once the answer is found, it will be used to compare with the answer from the one with increased further period demands in order to qualify for the discounts, or further discounts, and the better answer, justified by the objective of the algorithm, will be retained as the tentative answer to be used in the further comparison if there is any further level of discounts. Repeat such procedures until there are no more discount levels. Then a new order will be placed, and the order quantity will be determined by repeating the above procedures.

Least Unit Cost Method

This technique determines the order quantity based on the "unit cost" (total cost per unit) computed for each of the successive order quantities. The one with the local minimum unit cost will be chosen as the lot size of that order. The next order will be computed through identical procedures. When there are discounts available, the total cost will be modified by the sum of ordering cost and holding cost subtracting the discount saving. There will be two kinds of outcomes when a local minimum point is found. The first one is at the local minimum point the program checks to find whether the sum of demands has already reached the discount requirements. If it has, then the quantity determined from this point will be chosen as the ordering quantity. The second situation is the situation which occurs if the local minimum point is found with the summation of demands still below the discount requirements. It is possible, when we include further demands into the summation, that the unit cost will be lower because of the discount saving. So the first local minimum point is temporarily stored, and later compared with another candidate, the one with the quantity with discount savings. The candidate with a lower unit cost will be chosen as the desired ordering quantity. The logical steps of such searching is represented in the following flowchart (Figure 3-1).

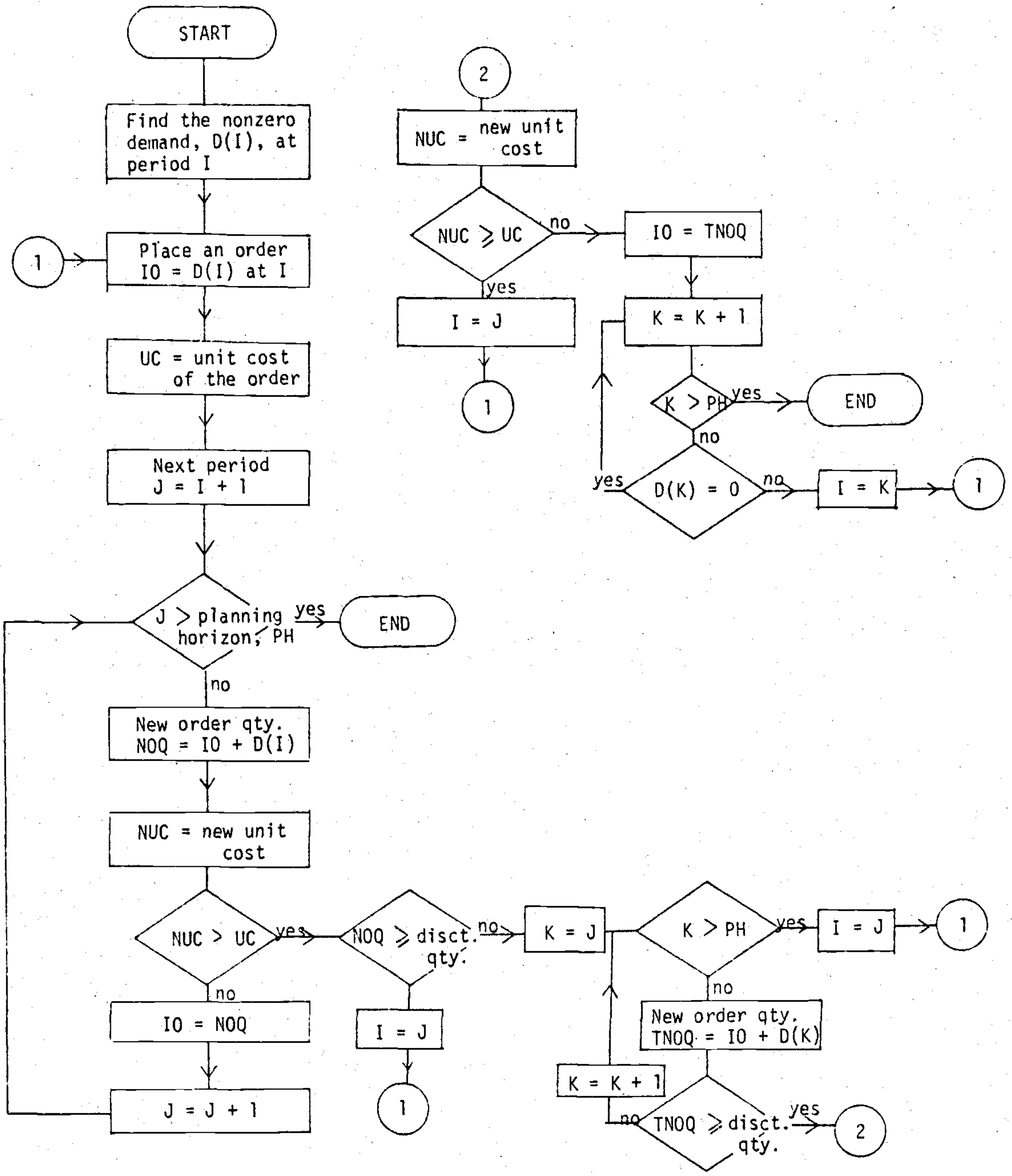


Figure 3-1 Flowchart of the Modified LUC Method

Numerical Example

Let's use the following example to explain the approach. Assume a manufacturing operation where the set-up cost is $92.00 per order and the holding cost is $2.00 per unit-period. The inventory holding cost is based on the ending inventory, and no split order is allowed. The required order quantity to get a discount is 200 units, the discount rate is $1/unit. The periodic demands within the planning horizon are:

Period	1	2	3	4	5	6	7	8	9	10	11	12
Demand	50	80	180	80	0	0	180	150	10	100	180	95

The search and computations of this example using the Lease Unit Cost Method are shown in Figure 3-2.

Silver-Meal Method

The method of Silver-Meal represents another approach to searching for the right ordering quantity at the right time. The Silver-Meal method finds an ordering quantity that leads to a local minimum cost per period. Therefore, starting with the first unfulfilled demand, the following demands are added onto it, and at each time the total cost, which is the ordering cost plus the holding cost subtracting the discount saving, will be found along with the number of periods involved. The total cost per period at each time is used to compare with the previous values until the local minimum point is found. Again, as in the case of "least unit cost," two kinds of outcomes may occur. One with the sum of demands at the local minimum point already satisfy-

Demands	50	80	180	80	0	0	180	150	10	100	180	95
Total Cost	92	252	1692 -310 =1382				92	392 -330 =62	432 -340 =92			
Total Qty.	50	130	310				180	330	340			
Unit Cost	1.84	1.94	4.46				0.51	0.19	0.27			
Local Min.	*							*				
Next Total Qty > Disct. Qty	no	yes						yes				
Total Cost		92	452 -260 =192	772 -340 =432					92	292	1012 -290 =722	1582 -385 =1197
Total Qty.		80	260	340					10	110	290	385
Unit Cost		1.15	0.74	1.27					9.20	2.65	2.49	3.11
Local Min.			*								*	
Next Total Qty > Disct. Qty			yes								yes	
Total Cost				92	92	92	1172 -260 =912					92
Total Qty.				80	80	80	260					95
Unit Cost				1.15	1.15	1.15	3.50					0.97
Local Min.						*						*
Next Total Qty > Disct. Qty						yes						-
To Order:	50	260	-	80	-	-	330	-	290	-	-	95

Total: Ordering Cost = 552 Holding Cost = 1580 Discount Saving = 880 Net Cost = 1252

Figure 3-2 An Example of LUC Method in the Discount Situation

ing the discount requirement, in which case that sum will be chosen as the ordering quantity. The other one represents the case that the sum is below the discount requirement. Then the sum will be stored, and compared with the situation when further demands are added into the sum to get the discount advantages, and the better one, based on the criteria of lower total cost per period, will be chosen as the desired ordering quantity. The logical steps of this approach are given in the flowchart of Figure 3-3.

Inoue-Chang Method

A third approach to solving the dynamic lot sizing problem under the discount situation is extended from the Inoue-Chang Method. Basically speaking, the Inoue-Chang Method decides whether or not to place an order at each period based upon comparison of the ordering cost and the holding cost. Starting with the first demand, the method places an order. The next scheduled order is tentatively set at the period where the holding cost becomes larger than the ordering cost. The method then backtracks and checks all other alternatives within the time interval, and determines tentatively how many periods should be covered by that scheduled order. Since the availability of discounts may induce certain significant saving, every time the order is tentatively scheduled a check will be carried out to see if the order's quantity has already brought in the discount advantages. If the answer is "yes", the method will start placing the next order; otherwise, the tentatively scheduled order will extend its coverage to further

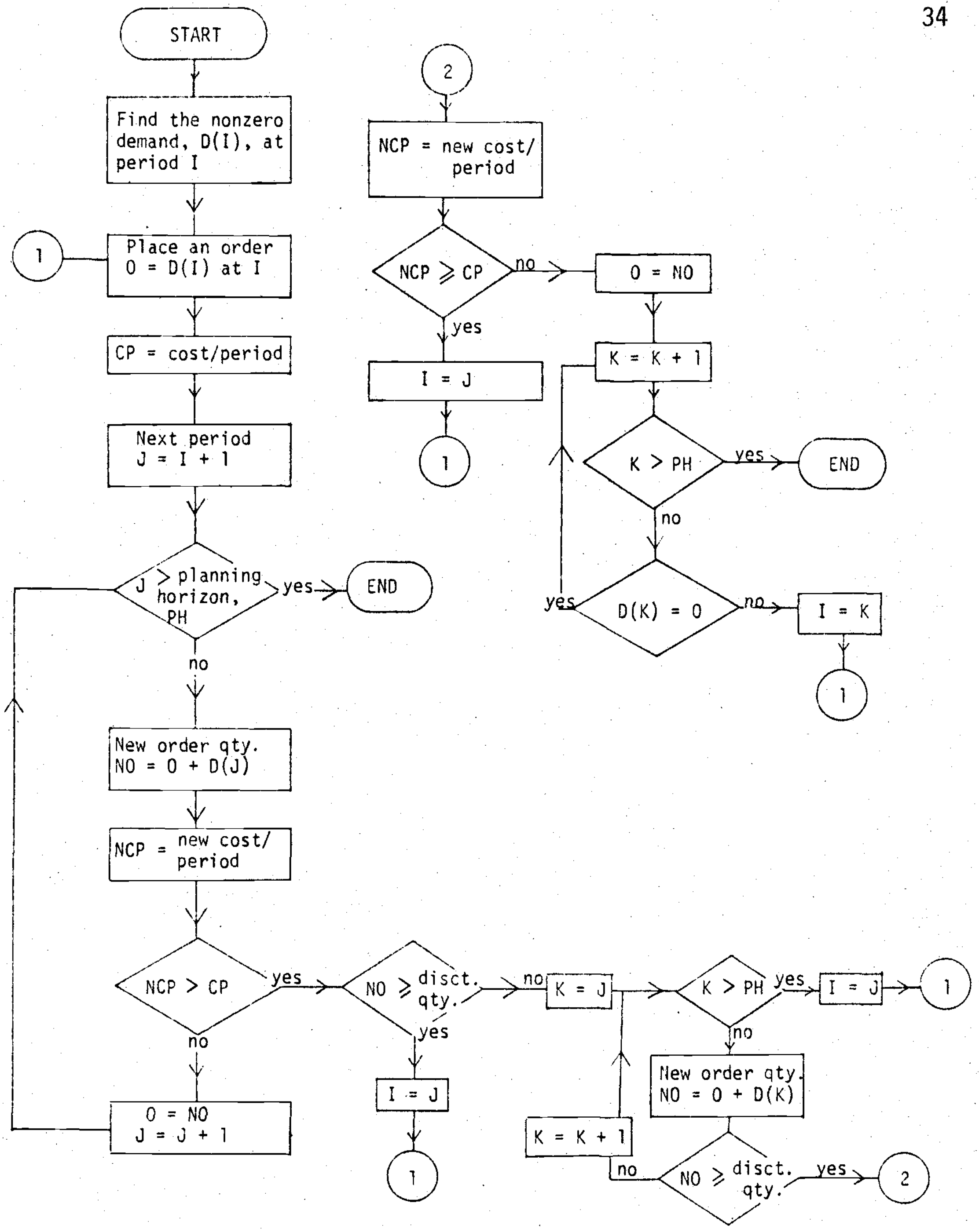


Figure 3-3 Flowchart of the Modified Silver-Meal Method

Demands	50	80	180	80	0	0	180	150	10	100	180	95
Total Cost	92	252	1692 -310 =1382				92	392 -330 =62	432 -340 =92	1032 -440 =592		
Total Periods	1	2	3				1	2	3	4		
Cost/Period	92	126	461				92	31	30.7	148		
Local Min	*								*			
Next Total Qty > Disct. Qty	no	yes							yes			
Total Cost		92	452 -260 =192							92	452 -280 =172	832 -375 =457
Total Periods		1	2							1	2	3
Cost/Period		92	96							92	86	152.3
Local Min		*									*	
Next Total Qty > Disct. Qty		yes									yes	
Total Cost			92	252 -260 =-8	252 -260 =-8	252 -260 =-8	1692 -440 =1252					92
Total Periods			1	2	3	4	5					1
Cost/Period			92	-4	-2.7	-2	250.4					92
Local Min						*						*
Next Total Qty > Disct. Qty						yes						-
To Order:	50	80	260	-	-	-	340	-	-	280	-	95

Total: Ordering Cost = 552 Holding Cost = 860 Discount Saving = 880 Net Cost = 532

Figure 3-4 An Example of Silver-Meal Method in the Discount Situation

demands to reach the discount requirements. In this way, the method will bring the saving from the discount and share the ordering cost with more demands while incurring an increased holding cost. A comparison of increased holding cost to the saving will tell whether such extension of coverage is desirable. The method will repeat these procedures until the end of the planning horizon. A backward search representing phase-2 of this method is used to check any improvement that can be made by moving an order within the time interval to combine to a previous order. This backward search is little different from the one used for the no-discount situation in which we are aware of the trade-off of holding costs by assigning a demand to the different orders. When the discount is available, the situation is complicated. Moving out a demand from one order to its prior order may cause a change in the order quantity to satisfy the discount requirements. In order to avoid these complications, it is proposed to check only the possibility of saving from combining one order to its prior order.

Let's consider two consecutive orders. Each one may already reach the discount requirements or it may not. Therefore, the total will be four situations. The two with the first order less than discount requirements can be neglected, because such check was made during the forward search, in which when one order was less than the discount requirement, the search extended the order coverage to further demands to test the possibility of getting saving from discounts. When the first order is greater than the discount requirements, and the second order is also greater than the discount requirements, further

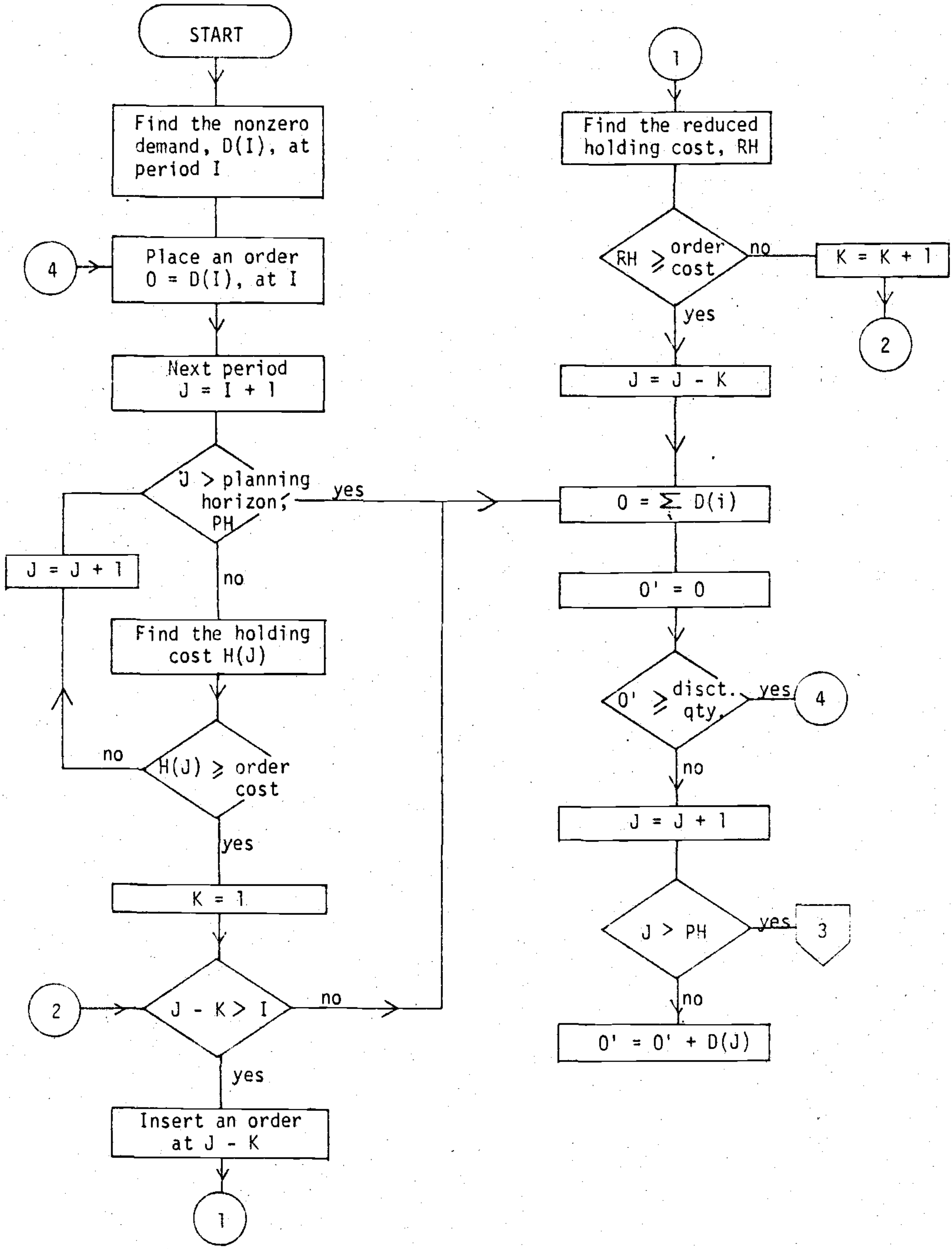
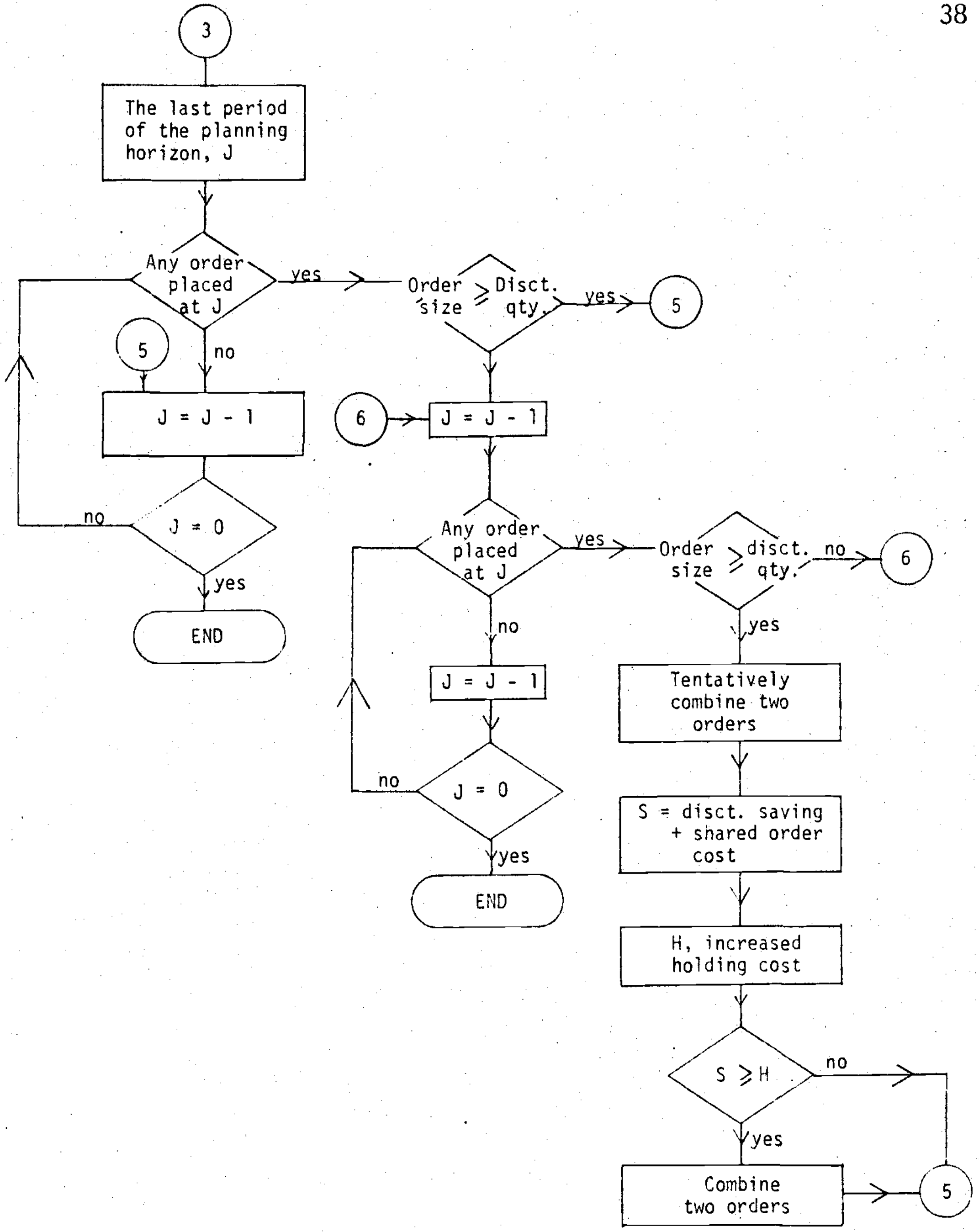


Figure 3-5 Flowchart of the Modified Inoue-Chang Method



Flowchart of the Modified Inoue-Chang Method (Continued)

Phase - 1:

Demands	50	80	180	80	0	0	180	150	10	100	180	95
Tentative Order	*											
Holding Cost	0	160	880									
> Ordering Cost		yes										
Order Qty. > Disct. Qty.	no	no	yes									
Total Saving[1]			387.2									
Incre. Holding Cost > Saving			yes									
Tentative Order		*							*			
Holding Cost		0	360						0	200	920	
> Ordering Cost			yes							yes		
Order Qty. > Disct. Qty.		no	yes						no	no	yes	
Total Saving			323.7								98.8	
Incre. Holding Cost > Saving			yes								yes	
Tentative Order			*							*		
Holding Cost			0	160						0	360	
> Ordering Cost				yes							yes	
Order Qty. > Disct. Qty.			no	yes						no	yes	
Total Saving				288.3							339.1	
Incre. Holding Cost > Saving				no							yes	
Tentative Order							*				*	
Holding Cost							0	300			0	190
> Ordering Cost								yes				yes
Order Qty. > Disct. Qty.							no	yes			no	yes
Total Saving								371.8				306.8
Incre. Holding Cost > Saving								no				no
To Order:	50	80	260	-	-	-	330	-	10	100	275	-

Total: Ordering Cost = 644 Holding Cost = 650 Discount Saving = 865 Net Cost = 429

1: Total Saving = Disct. Saving + Ordering Cost Shared by the Additional Qty.

Figure 3-6 An Example of Inoue-Chang Method in the Discount Situation

Phase - 2:

Demands	50	80	180	80	0	0	180	150	10	100	180	95
Decisions from Phase - 1 to Order:	50	80	260				330		10	100	275	
Order without Disct. while Its Proior Order Has Disct.	no	no	no				no		yes	no	no	
Combine with the Prior Order:												
Incre. Holding Cost									40			
Ordering Cost Saving									-92			
Discount Saving									-10			
Net Cost									-62			
To Order:	50	80	260	-	-	-	340	-	-	100	275	-

Total: Ordering Cost = 552 Holding Cost = 690 Discount Saving = 875 Net Cost = 367

An Example of Inoue-Chang Method in the Discount Situation (Continued) tests can also be neglected during the backtrack search. This is because the combination of those two orders will simply raise the holding cost which is already greater than the ordering cost as we know from the forward search. The only situation that requires the backtrack check is the situation in which the first order is greater than the discount requirement while the second order is not. The combination of these two orders will save one ordering cost, and some discount saving from the later order while increasing the holding cost by covering the later demands in an earlier order. Again, a comparison of the increased holding cost and savings will tell the decision-maker whether or not to combine his orders.

Dynamic Programming Approach

In the Chapter 2, the dynamic programming approach has been in detail discussed. In a discount situation, the total cost is extended to include the ordering cost plus the holding cost subtracting the discount saving. The exhaustive search can be reduced through the implementation of Planning Horizon Theorem. The modification of that theorem to adapt a discount environment is also proposed in Chapter 2. The same example to explain the other heuristic approaches is used for the dynamic programming's approach in Figure 3-7. The shadow part represents the part of calculations that can be saved from the modified planning horizon theorem in discount situation derivated at Chapter 2. Again, since this approach guarantees an optimal solution, it is used as a benchmark to test the performance of other heuristic approaches.

	1	2	3	4	5	6	7	8	9	10	11	12
50	92*											
80	252	184*										
180	972 -310 =662	544 -260 =284	276*									
80	1452 -390 =762	864 -340 =524	436 -260 =176*	368								
0	762	524	176*	368	268							
0	762	524	176*	368	268	268						
180	3612 -570 =3042	2664 -520 =2144	1876 -440 =1436	1448 -260 =1188	988	628	268*					
150	5712 -720 =3042	4464 -670 =3794	3376 -590 =2786	2648 -410 =2238	1888 -330 =1558	1228 -330 =898	568 -330 =238*	360				
10	5872 -730 =5142	4604 -680 =3924	3496 -600 =2896	2748 -420 =2328	1968 -340 =1628	1288 -340 =948	608 -340 =268*	380	330			
100	7672 -830 =6842	6204 -780 =5424	4896 -700 =4196	3948 -520 =3428	2968 -440 =2528	2088 -440 =1648	1208 -440 =768	780 -260 =520	530	360*		
180	11272 -1010 =10262	9444 -960 =8484	7776 -880 =6896	6468 -700 =5748	5128 -620 =4508	3888 -620 =3268	2648 -620 =2028	1860 -440 =1420	1250 -290 =960	720 -280 =440*	452	
95	13362 -1105 =12257	11344 -1055 =10289	9486 -975 =8511	7988 -795 =7193	6458 -715 =5743	5028 -715 =4313	2743 -715 =2028	2620 -535 =2085	1820 -385 =1435	1100 -375 =725	642 -275 =367*	532
To Order:	50	80	260	-	-	-	340	-	-	100	275	-

Ø = 92
H = 2
Q = 200
S = 1

Figure 3-7 An Example of Wagner-Whitin Method in the Discount Situation

0/H	PH-1	PH-2	SM	LUC	WW
300/2	1541.52	1512.40	1620.62	1989.01	1364.57
206/2	1028.87	1003.12	1129.52	1486.39	914.05
120/2	522.83	511.76	587.98	933.00	454.51
92/2	325.41	320.39	387.12	702.96	286.69
48/2	21.11	19.72	63.36	240.90	-4.94

Table 3-7. Comparison of Average Costs When Discounts Are Available, Discount Rate = $1/Unit

0/H	PH-1	PH-2	SM	LUC	WW
300/2	624.80	575.00	835.54	1071.18	394.34
206/2	162.90	115.34	420.68	597.82	-24.56
120/2	-321.50	-353.88	-107.62	115.32	-446.14
92/2	-483.90	-507.68	-298.20	-73.10	-594.50
48/2	-736.50	-748.96	-560.82	-373.72	-844.32

Table 3-8. Comparison of Average Costs When Discounts Are Available, Discount Rate = $2/Unit

0/H	PH-1	PH-2	SM	LUC	WW
300/2	-342.04	-427.37	-43.13	109.39	-641.97
206/2	-784.10	-864.54	-411.55	-327.19	-1039.19
120/2	-1231.49	-1292.16	-828.23	-789.22	-1426.53
92/2	-1385.18	-1435.63	-979.24	-930.54	-1562.28
48/2	-1649.03	-1680.95	-1230.66	-1193.42	-1790.46

Table 3-9. Comparison of Average Costs When Discounts Are Available, Discount Rate = $3/Unit

0/H	PH-1	PH-2	SM	LUC	WW
300/2	-1328.66	-1456.04	-868.38	-900.08	-1700.86
206/2	-1751.60	-1885.64	-1263.86	-1313.90	-2090.32
120/2	-2185.62	-2292.30	-1674.98	-1691.24	-2459.28
92/2	-2338.42	-2429.22	-1803.82	-1813.98	-2585.58
48/2	-2583.12	-2650.40	-2036.10	-2073.36	-2794.58

Table 3-10. Comparison of Average Cost When Discounts Are Available, Discount Rate = $4/Unit

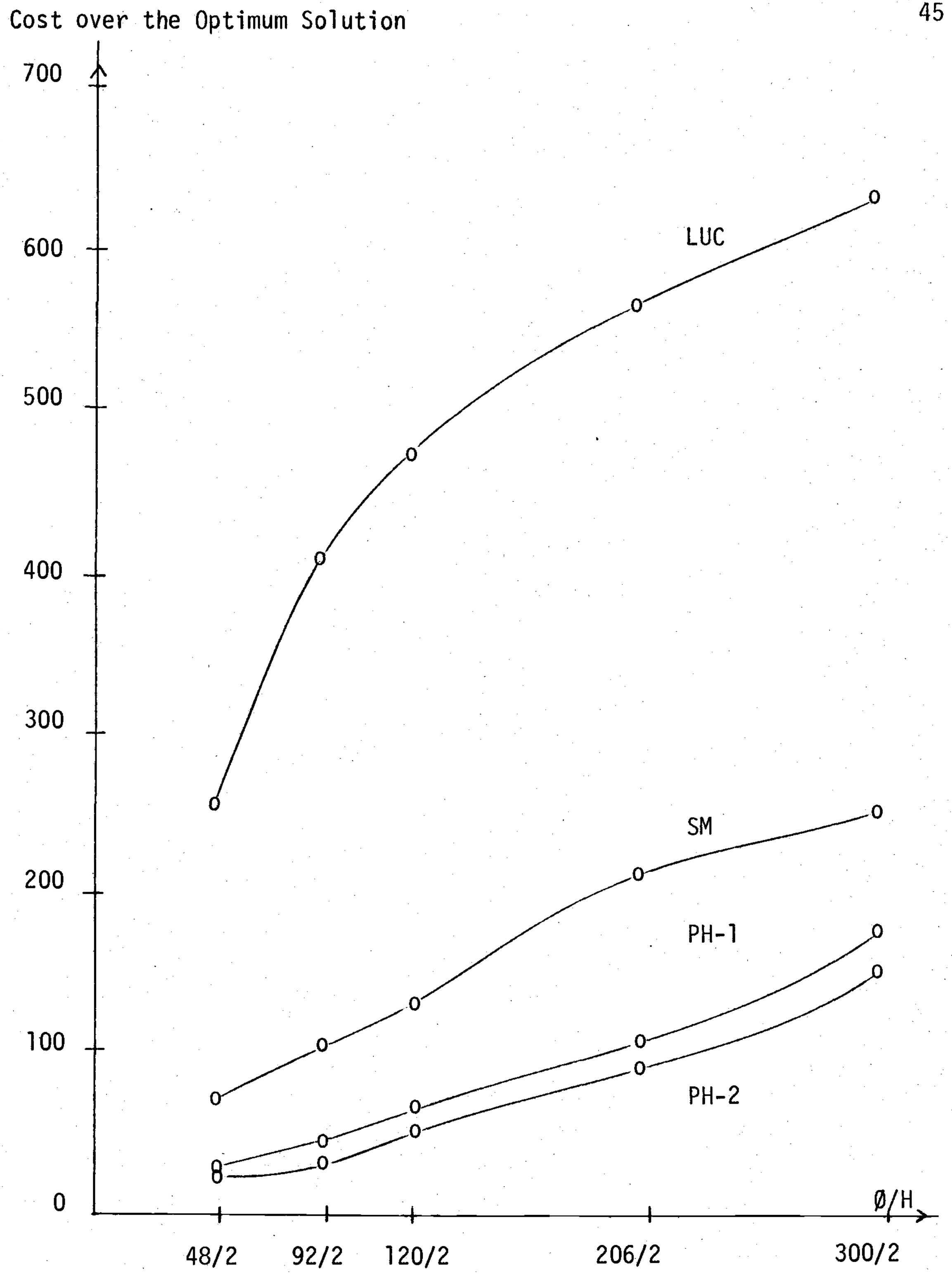


Figure 3-8 Comparison of Average Cost at Discount Situation, Discount Rate = $1/Unit

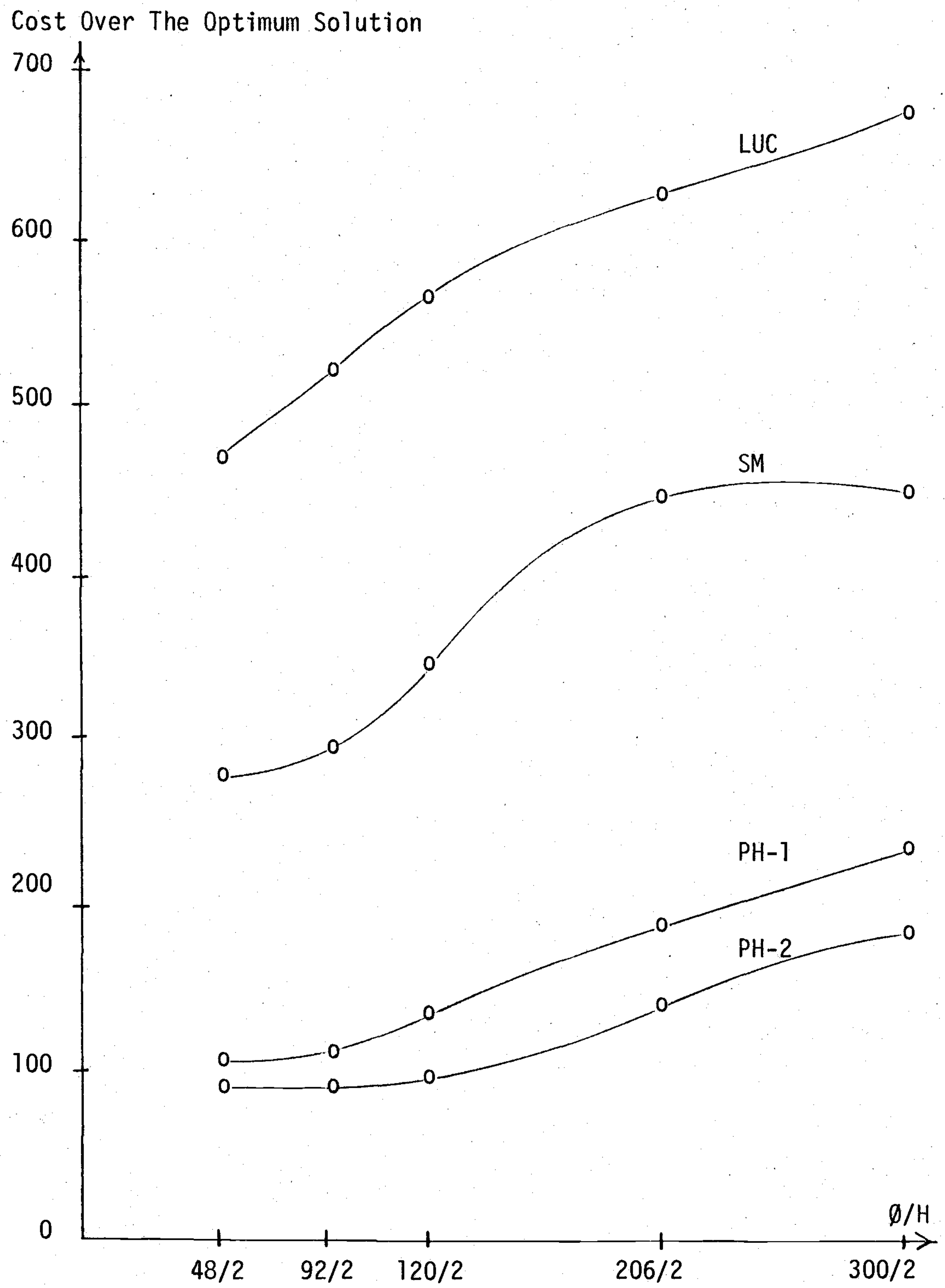


Figure 3-9 Comparison of Average Cost at Discount Situation, Discount Rate = $2/Unit

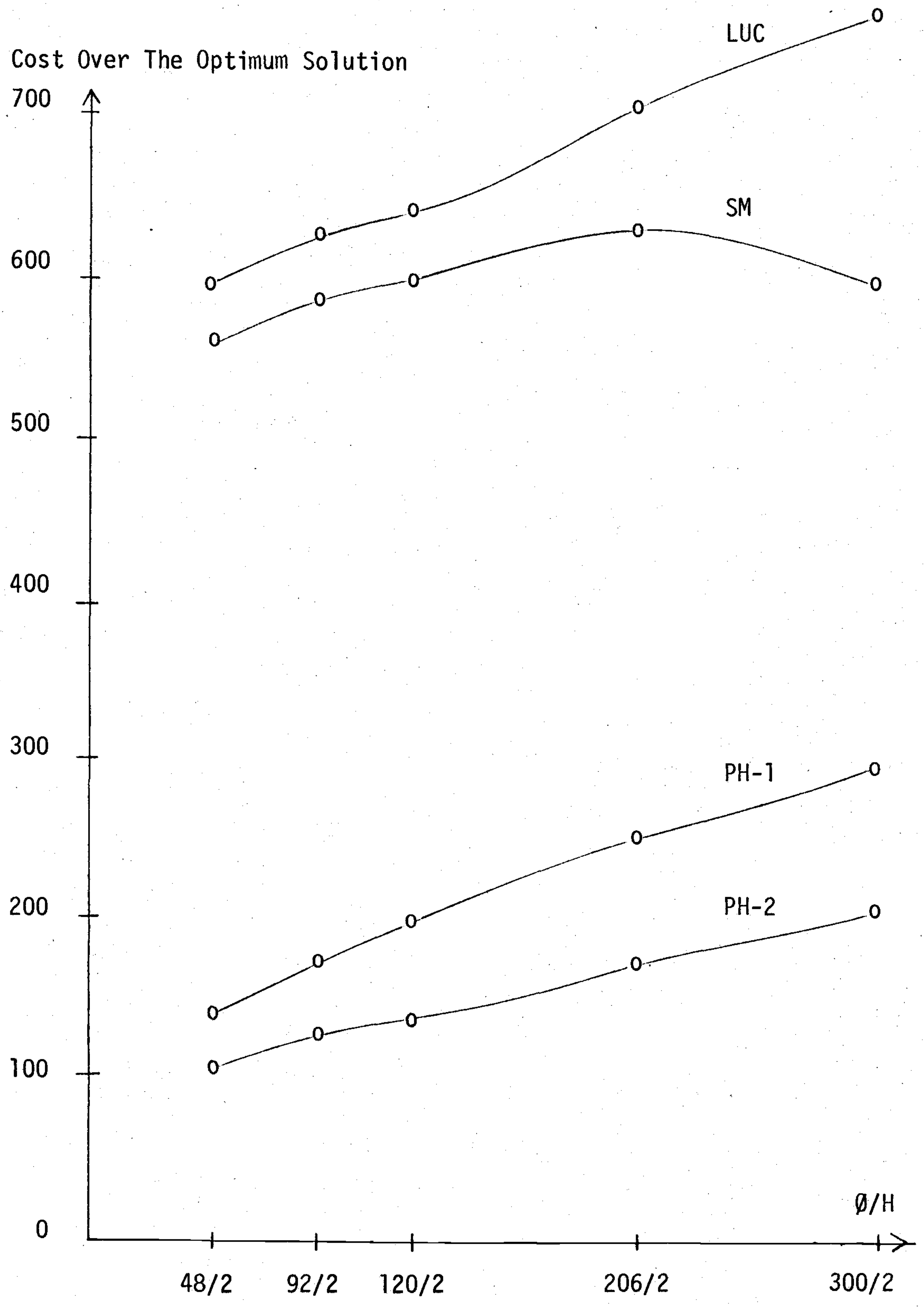


Figure 3-10 Comparison of Average Cost at Discount Situation, Discount Rate = $3/Unit

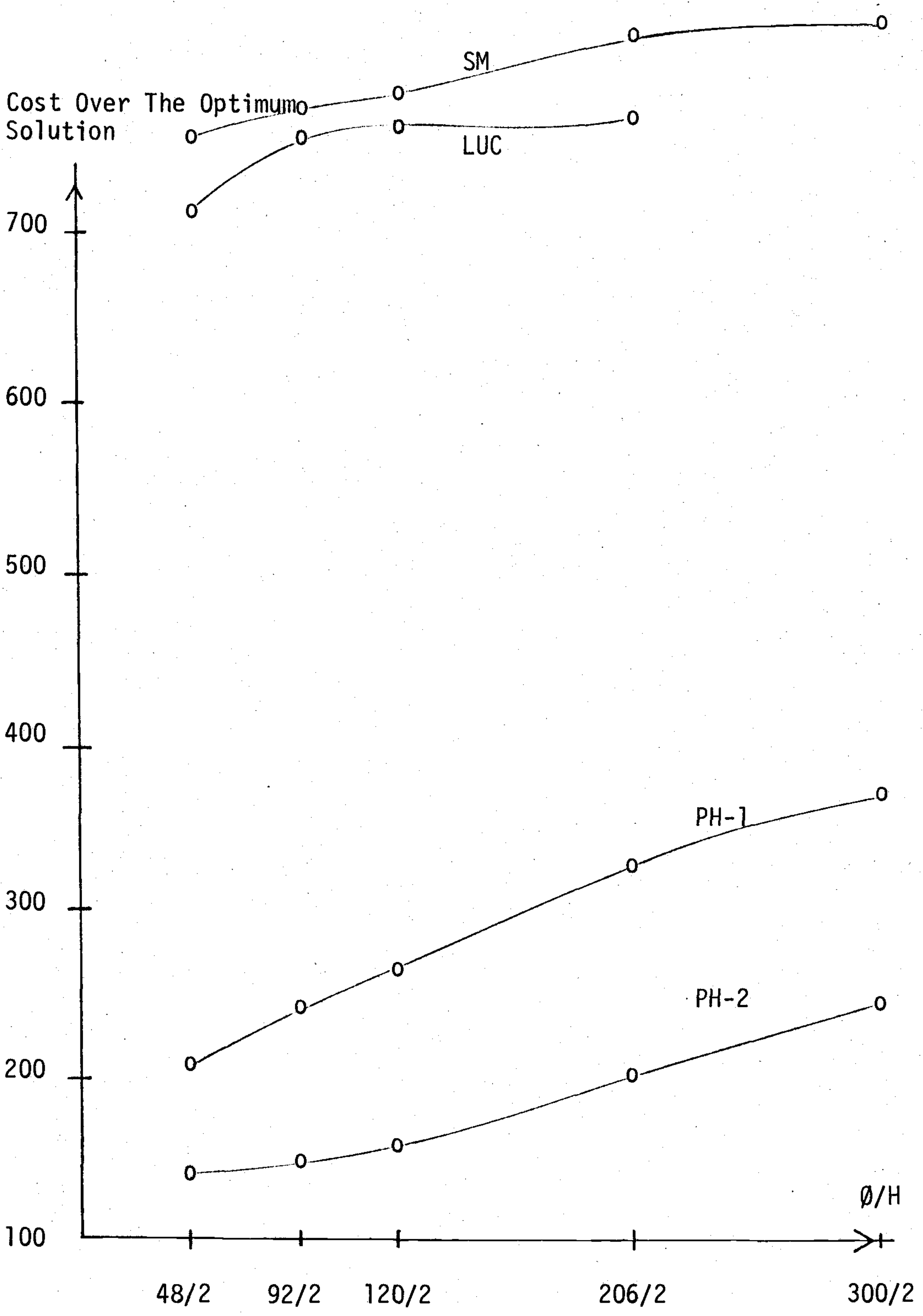


Figure 3-11 Comparison ov Average Cost at Discount Situation, Discount Rate = $4/Unit

Some Comparisons

The same data that have been generated to test the performance of different approaches in a no-discount situation are used to test the situations with discounts available. The different discount rates are chosen as a function of the holding cost. Since in order to cover more demands in an order to get discounts, the first trade-off is the increased holding cost. Therefore, the discount rate is set to 50%, 100%, 150%, and 200% of the value of the holding cost. Some testing results are listed at Table 3-7 to Table 3-10, and Figure 3-8 to Figure 3-11.

CHAPTER IV

JOINT ORDER MULTIPERIOD MULTIPLE-ITEM DYNAMIC LOT SIZING PROBLEM WHEN DISCOUNTS ARE AVAILABLE

Introduction

The multiperiod dynamic lot sizing problem is often difficult because the demands are varying with time and no general mathematical function is usually known to describe such demands. The complexity increases when the discounts are available. Adding further to the difficulty is when the multiple-item instead of single-item is under consideration. This type of problem has not been researched before. The objective of this work is to develop different heuristic programs and optimal algorithms, based on the assumptions mentioned in Chapter I, to search for the optimal and near-optimal solutions using different approaches for the joint order multiperiod, multiple-item dynamic lot sizing problems when the discounts are available. The heuristic programs will basically be the extensions of Least Unit Cost Method, Silver-Meal Method, and Inoue-Chang Method, as they represent three different approaches to solve the problems. The optimal algorithm will use the dynamic programming approach to search for the optimal solution among all the possible feasible solutions. In order to reduce the huge number of searches in the multiple-item discount situations, the extension of planning horizon theorem in such complex problem is discussed and used along the search.

Heuristic Program 1

No Discount Situation

This program is developed from the single item "Least Unit Cost" method. The criterion of this program is to select the ordering quantity which will lead to a local minimum unit cost. Since the problem it will deal with is a joint order multiple-item dynamic lot sizing problem, the nature of the interaction among these items must first be considered. When a period's demands are covered by the scheduled receipt, it is possible to cover only a small number of item's demand. Thus, the priority of items' demands to be covered by the scheduled receipt should be studied. In order to avoid too much complication, let's start with a multiple-item dynamic lot sizing problem without the discounts, and assume the holding cost to be a constant value for all items.

Theorem 4-1: Whether a period's demands should be covered by a scheduled order depends only on the previous demands covered by the order, the holding cost, the ordering cost, and the time length (number of periods) away from the scheduled order's time, and is regardless of the current demand quantities.

Proof: Let's assume there are m items involved, i^{th} item's demand at t^{th} period is denoted by D_{ti}. Suppose the unit cost of a scheduled order covering up to $(t-1)^{th}$ period is R_0, and the unit cost changes to R_A after a certain combination of demands at t^{th} period, denoted by D_A, is added to the order. The decision to add D_A to the order or not is based upon the comparison of R_o to R_A.

Let C_{t-1} represent the total cost of the order before the D_A is added, I_{t-1} represent the total number of items in the order, H represent the holding cost per unit per period, and assume all the items to have the same holding cost. Also, let L represent the number of periods those t^{th} period's demand away from the period to place the order. Then

$$R_o = C_{t-1}/I_{t-1}$$

and $$R_A = [C_{t-1} + D_A LH] / [I_{t-1} + D_A]$$

and the comparison of which is larger can be performed as follows:

$$R_o : R_A$$

$$C_{t-1} / I_{t-1} : [C_{t-1} + D_A LH] / [I_{t-1} + D_A]$$

$$C_{t-1} I_{t-1} + C_{t-1} D_A : C_{t-1} I_{t-1} + D_A LHI_{t-1}$$

$$C_{t-1} : LHI_{t-1}$$

This concludes that the study of whether the value of R_A is smaller than or equal to, or greater than R_o may depend upon the comparison of C_{t-1}, and LHI_{t-1} only.

Theorem 4-2: Whenever a demand D_{ti} is found to lower the unit cost of an order when it is included in that order, all the other demands D_{tj}, $j \neq i$, should also be covered by that order.

Proof: Let's use R_A and R_{A+B} to represent two unit cost ratios including two combinations of certain t^{th} period's demands. D_{A+B} is D_A plus some other demands, D_B, in the same t^{th} period, C_{t-1} represents the total cost of the order before including any demand from t^{th}

period, I_{t-1} is the total number of units in the order before including any t^{th} demand, $R_A = [C_{t-1} + D_A LH] / [I_{t-1} + D_A]$ and $R_{A+B} = [C_{t-1} + D_A LH + D_B LH] / [I_{t-1} + D_A + D_B]$, where $D_A \neq 0$, $D_B \neq 0$.

The comparison of $R_A : R_{A+B}$ will have

$$I_{t-1}C_{t-1} + I_{t-1}D_A LH + D_A C_{t-1} + D_A D_A LH + D_B C_{t-1} + D_B D_A LH : I_{t-1}C_{t-1} + I_{t-1}D_A LH + I_{t-1}D_B LH + D_A C_{t-1} + D_A D_A LH + D_A D_B LH$$

which gives $D_B C_{t-1} : I_{t-1} D_B LH$

when $D_B \neq 0$, $\quad C_{t-1} : I_{t-1} LH$

which means, if $R_A < R_o$ then we have $R_{A+B} < R_A < R_o$, and on the other side, if $R_A > R_o$, then $R_{A+B} > R_A > R_o$. That concludes the proof.

One question may arise, how about if $R_A < R_o$ while R_B is found to be greater than R_o; will such situation lead to the contradict conclusion: $R_A < R_o$ implies $R_{A+B} < R_A < R_o$ and $R_B > R_o$ will imply $R_{A+B} > R_B > R_o$? The answer is that such situation will not happen because whether R_B is greater than R_o, or R_A is smaller than R_o, they depend only upon the comparison of C_{t-1} and LHI_{t-1}. If $LHI_{t-1} < C_{t-1}$, both R_A and R_B will be less than R_o, and vice versa. The situation of different items with different holding costs will be somewhat complicated. Let's consider a single demand D_{ti} with the holding cost H_i, which is being added to the scheduled order. This leads to the ratio R_i / R_o:

$$R_i / R_o = \left\{ [C_{t-1} + D_{ti} LH_i] / [I_{t-1} + D_{ti}] \right\} / [C_{t-1} / I_{t-1}] = [1 + D_{ti}(LH_i / C_{t-1})] / [1 + D_{ti}(1/I_{t-1})]$$

Therefore, the comparison of $I_{t-1}LH_i$ / C_{t-1} will tell which unit cost, R_i or R_o, is smaller. But this time H_i is not a constant for all the items. Some lower holding cost will lead $LH_jI_{t-1}<C_{t-1}$, which implies $R_j<R_o$, and other higher holding cost may lend $LH_kI_{t-1}>C_{t-1}$, which implies $R_k>R_o$. Also, when we consider to add D_m into the order, then compare with to add $D_m + D_n$ into the order, the two unit costs, R_m and R_{m+n}, will have the following ratio:

$$R_{m+n} / R_m = \left\{[C_{t-1} + D_m LH_m + D_n LH_n] / [I_{t-1} + D_m + D_n]\right\}$$

$$/ [(C_{t-1} + D_m LH_m) / (I_{t-1} + D_m)] = [1 + D_n(LH_n / C^1_{t-1})] /$$

$$[1 + D_n(1 / I^1_{t-1})]$$

Where $C^1_{t-1} = C_{t-1} + D_mLH_m$, $I^1_{t-1} = I_{t-1} + D_m$. Therefore, studying the ratio, R_{m+n} / R_m, is equivalent to considering:

$$LH_n(I_{t-1} + D_m) : (C_{t-1} + D_mLH_m)$$

$$LH_nI_{t-1} + LH_nD_m : C_{t-1} + D_mLH_m$$

if $H_n>H_m$ and if we have $LH_nI_{t-1}>C_{t-1}$ and $LH_nD_m>LH_mD_m$, then the ratio will be greater than 1. That tells if all the holding costs are listed in the ascending sequence, starting with the lowest holding cost and search upward until an H_j is found that $I_{t-1}LH_j>C_{t-1}$, we can neglect all the items since the j^{th} one, because our adding them to the order will raise the unit cost. Then how about those items with the holding cost H_k such that $I_{t-1}LH_k<C_{t-1}$? Suppose K^{th} item has the holding cost just lower than H_j to fulfill the requirement $I_{t-1}LH_k<C_{t-1}$, for all the other items with lower holding cost can lead to the following result:

If $H_i<H_k$ then $I_{t-1}LH_i<I_{t-1}LH_k<C_{t-1}$ and $H_i<H_k$ that will imply:

$$LH_iI_{t-1} + LH_iD_m<C_{t-1} + D_mLH_k$$

or $R_{k+i}<R_k$

If we further add item D_w into the order, and $H_w<H_i<H_k$,

$$R_{k+i+w} / R_{k+i} = [1 + D_w(LH_w / C''_{t-1})] / [1 + D_w(1/I''_{t-1})]$$

where $C''_{t-1} = C_{t-1} + D_kLH_k + D_iLH_i$

$$I''_{t-1} = I_{t-1} + D_k + D_i$$

Again, we can determine whether R_{k+i+w} is greater or smaller than R_{k+i} by looking at the ratio:

$$LH_w(I_{t-1} + D_k + D_i) / (C_{t-1} + D_kLH_k + D_iLH_i).$$

Since $H_w<H_i<H_k$ and $I_{t-1}LH_w<I_{t-1}LH_i<I_{t-1}LH_k<C_{t-1}$, we know that $R_{k+i+w}<R_{k+i}$. That tells the following theorem.

Theorem 4-3: When all the items do not have a constant holding cost, at each period, it is the item's holding cost that determines whether the item's demand shall be added to the order to lower the unit cost. Therefore, at t^{th} period, we may add all the items with the holding cost H_i, such that $I_{t-1}LH_i<C_{t-1}$, to the order, and that will lower the unit cost of the order.

Following such procedure we will be able to determine which item should be included in the scheduled order. But since our assumption is to have no backlog and there is no capacity constraints, there does not seem to be any reason to include a partial number of demands of a period into an order while leaving the rest to start a new order. Therefore, in a no-discount situation, we only think about whether all items' demands for a period should be included in the prior order, or we should start a new order. The procedure can use the highest holding cost, H_g, to make the test. If $I_{t-1}LH_g<C_{t-1}$, we add all the items into the prior order. If $I_{t-1}LH_g = C_{t-1}$, we still add all the items into the prior order because we know when the other items, with the lower holding cost, added into the order will lower down the unit cost. If $I_{t-1}LH_g>C_{t-1}$, the lowest holding cost, H_h, can be used to test whether a new order should be placed. If $I_{t-1}LH_h>C_{t-1}$, that means any addition of demands to the order will raise the unit cost. The $I_{t-1}LH_h = C_{t-1}$ also carries the information when all items are added to the order will raise the unit cost. If the result is $I_{t-1}LH_g>C_{t-1}$ while $I_{t-1}LH_h<C_{t-1}$, we can just add all the items' demands into the order and test if the result is in favor to keep the demands in the prior order or placing a new order.

Discount Situation

When discounts are available, the determination of an order's coverage of demands will extend to the possible discount saving through ordering some larger quantities. Since the problem we are dealing with is a multiple-item problem, each item may have its own discount rate for its required order quantity. Therefore, whenever a local minimum unit cost is reached, a check will be carried to see if every item has already received the discount advantage. If the finding is affirmative, the scheduled order will be considered as an appropriate one, and no other change will be made to that order. If the finding shows that an item's quantity is still below its discount-required quantity, there is a possibility of getting discount advantages through covering their subsequent demands in the order. Since the ordering cost is paid once for each order, to increase quantities will only result in the increase of holding cost, and the discount saving serves as a trade-off benefit. The comparison of the values for both sides will determine whether such increasing quantities are a desirable action. The same procedures may be carried out for the multiple discount level problems. If one item's demand in the order has already reached the requirement for a discount, there may exist some higher discount levels with a higher discount advantage. Whether we should extend the order's coverage to further demands for the higher discount saving or not depends upon the same comparison of increased holding cost versus the discount saving just like from no discount situation to the first discount situation.

Each time a series of demands for an item is added into the order to get discount advantages may affect the other item's demand that is being left behind. Let's use the following example to explain. Support we have K items, and the planning horizon is N period. When an order is placed at i^{th} period, it covers demand up to t^{th} period without considering the discounts. The $(t+1)^{th}$ period's demands are not included in the order because it is found they will raise up the unit cost. But after considering the discount advantages, the order's coverage zone crosses over certain item's $(t+1)^{th}$ demand, or even further. (Figure 4-1)

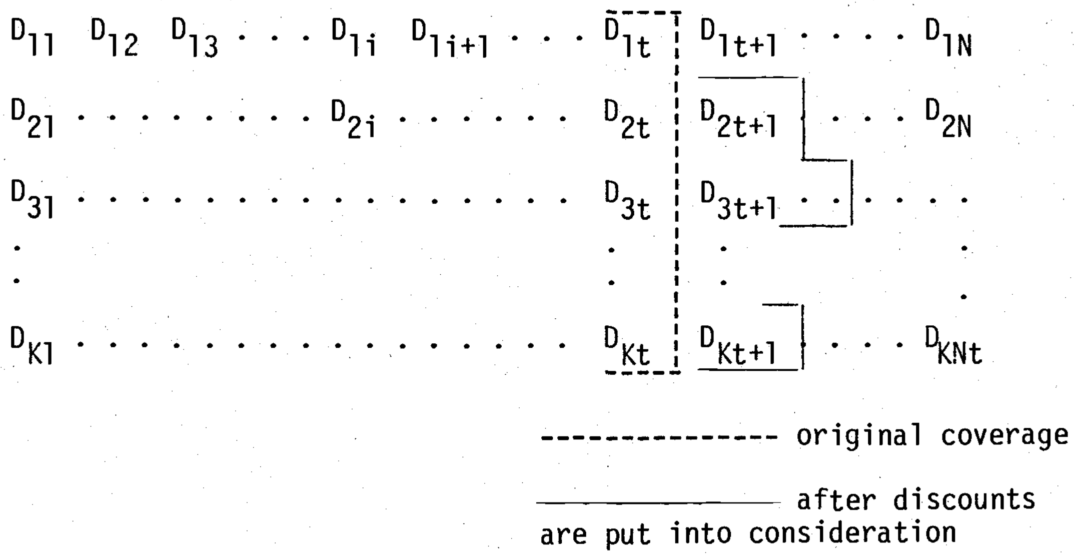

D_{11} D_{12} D_{13} . . . D_{1i} D_{1i+1} . . . D_{1t} D_{1t+1} . . . . D_{1N}
D_{21} . . . . D_{2i} . . . . D_{2t} D_{2t+1} . . . D_{2N}
D_{31} . . . . D_{3t} D_{3t+1} . . . . . .
D_{K1} . . . . D_{Kt} D_{Kt+1} . . . D_{KNt}
original coverage
after discounts are put into consideration

Figure 4-1 Order Coverage in a Discount Situation

The item 2, 3, and K are the examples that the order will extend its coverage to get discounts. We may find that the demands of $(t+1)^{th}$ period of item 1 and other items that are being left behind should be included in the prior order rather than to place another new order. Such results are difficult to predict; and once such demands are included in the order, they may well affect the later demands in the discount situation (unless every item reaches the highest discount rate). In order to avoid such cyclic effect, the program here only tests the discount advantage, and compares the ordering cost with the holding cost if the rest of the items for that period are also included in the order. This heuristic approach may not guarantee an optimal solution, but presents a relative simple method to read a feasible

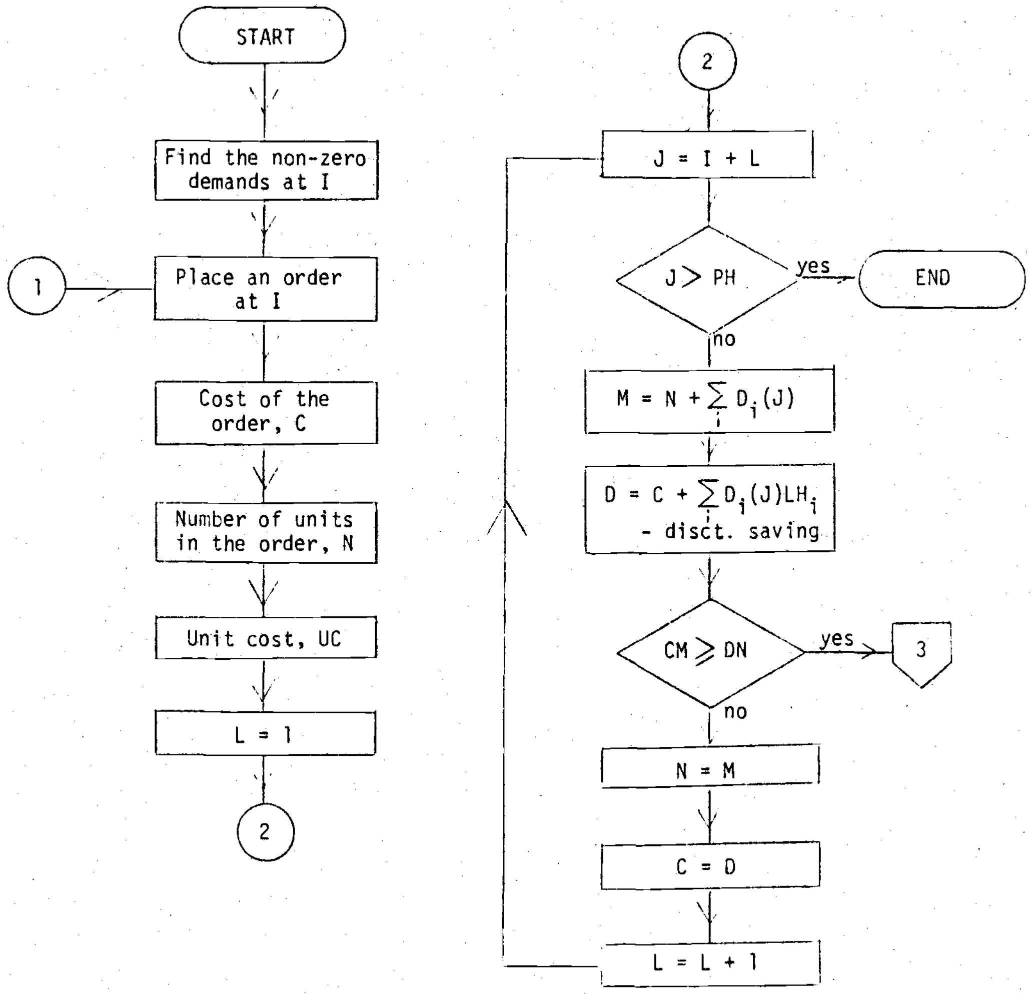
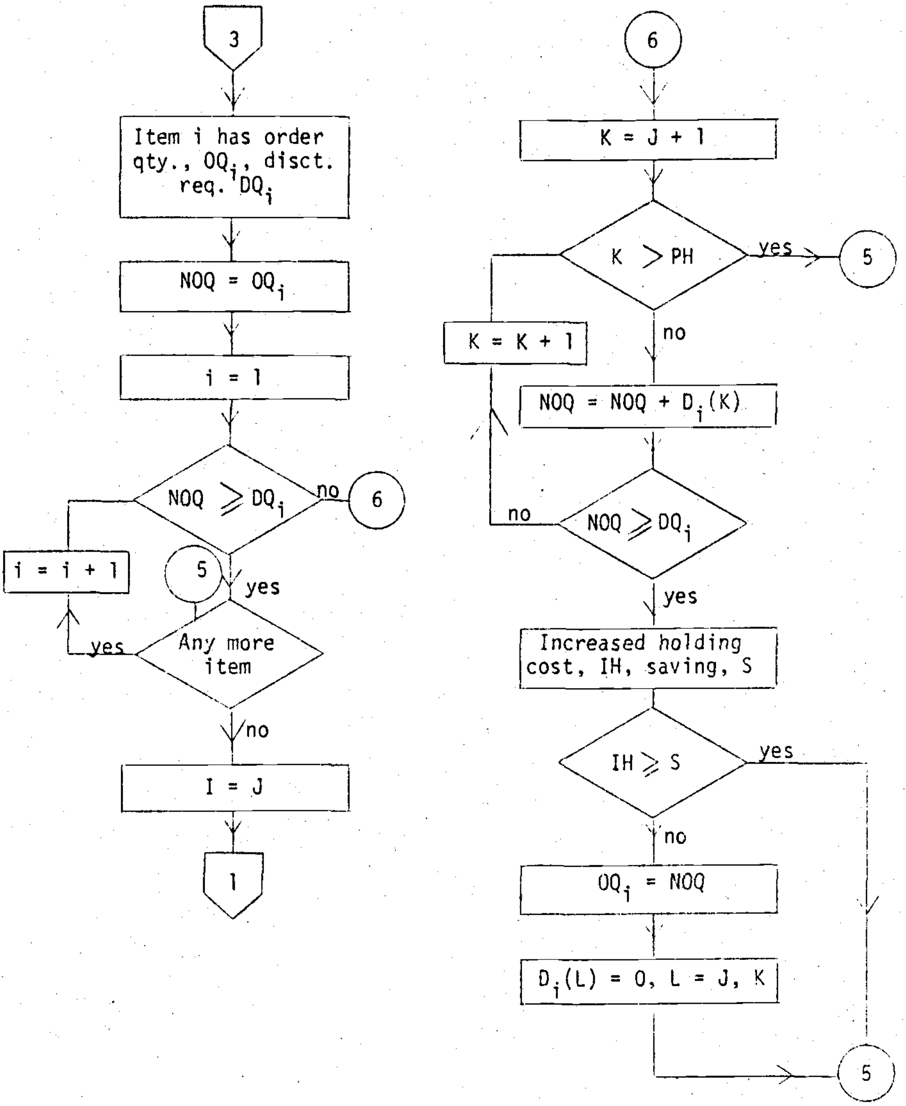


Figure 4-2 Flowchart of the Program-1



Flowchart of the Program-1 (Continued)

solution.

Heuristic Program 2

No Discount Situation

The criterion of this program is to select the ordering quantity which will lead to a local minimum total cost per period. Starting with the first non-zero demand, an order is placed in order to avoid any backlog. Since the measure of this approach is the total cost per period when an order extends its coverage to the subsequent period, the new total cost per period will be calculated and compared to the previous one. We may thus decide whether the demands of that period should be covered by the order.

Theorem 4-4: When the least total cost per period becomes the criterion to select the desired order quantity, the decision to include a period's demands into the scheduled order depends upon the comparison of the increased holding cost of that period to the prior total cost per period.

Proof: Suppose that an order is scheduled at i^{th} period and already includes the demand requirements from the i^{th} period up to the $(t-1)^{th}$ period, the number of periods involved in J-1. The total cost per period is:

$$R_o = \frac{C_{t-1}}{J-1}$$

Where C_{t-1} denotes the total cost (ordering and holding cost). When the order extends its coverage to the t^{th} period, the total cost will raise because of the increased holding cost and the new total cost per period,

$$R_A = [C_{t-1} + \sum_{i=1}^{M} (J-1)\ H_i\ D_{ti}] / J$$

Therefore, the comparison of two ratios will be:

$$R_A : R_o \quad \text{which gives}$$

$$[C_{t-1} + \sum_{i=1}^{M} (J-1)\ H_i D_{ti}](J-1) : J\ C_{t-1}$$

$$[\ \sum_{i=1}^{M} (J-1) H_i D_{ti}](J-1) : C_{t-1}$$

$$\sum_{i=1}^{M} (J-1) H_i D_{ti} : C_{t-1} / (J-1) = R_o$$

The left hand side stands for the increased holding cost from the M items demand at t^th period, and that concludes the proof.

From the above theorem, it is found that comparing a period's demands' holding cost to the previous total cost per period can determine whether the new total cost per period is up or down. In case there is any interest in the priority to choose one item's demand into the order, that priority will depend upon that item's holding cost only, since for each individual's demand added into the order, the resultant total cost per period, R_A, will have the similar outcome as

$$R_A : R_o \quad \text{we will have} \quad (J-1) H_i D_{ti} \ : \ R_o$$

where J-1 is a fixed value, and the comparison depends upon that item's current period's holding cost. The effect of adding other item's demands into the order is just to aggregate the total holding cost and use it to compare with the existing criterion's measure, the total cost per period at the last period. When there is no discount involved, the order will cover each period's demands either completely, or it will not cover any of them at all (here assume a zero-demand is also called a demand). The reason is that backlogging is not allowed in the problem. Any unfilled demand requires a new order. Therefore part of the demands being covered by the prior order will only carry a heavier holding cost than if they are covered by the new order.

In a special case when there is only one item involved, the procedure will follow the same steps to determine whether it is desirable to include the demand of the period in the scheduled order, but the comparison each time will be that single item's holding cost to R_o:

$$(J-1)HD_t : R_o = C_{t-1} / (J-1)$$

or $$(J-1)^2 HD_t : C_{t-1} = 0 + H \sum_{i=1}^{J-1} (i-1) D_{Ls+i}$$

which is the same expression derived by Silver and Meal for a single-item problem, and Ls represents the end period of last order (Silver and Meal, 1973).

Discount Situation

When discounts are available, the cost function will involve the saving of discounts. When an order is placed at first period, and it contains item's demands up to t^{th} periods, the cost function will be:

$$f(t) = 0 + \sum_{i=1}^{K} \sum_{j=1}^{t} h_i (j-1) D_{ji} - \sum_{i=1}^{K} \left(\sum_{j=1}^{t} D_{ji} \right) G_i \left(\sum_{j=1}^{t} D_{ji} \right)$$

where G_i represents the discount function, and its value depends on the order quantity only.

Again, for each subsequent period, the total cost per period will be compared with the value at its prior period in order to determine whether the period's demand should be included in the order.

$$R_A = f(t) / t : f(t-1) / (t-1) = R_o$$

which gives:

$$(t-1)\, f(t) : t f(t-1)$$

$$(t-1) \left[\emptyset + \sum_{i=1}^{K} \sum_{j=1}^{t} h_i (j-1) D_{ji} - \sum_{i=1}^{K} \left(\sum_{j=1}^{t} D_{ji}\right) G_i\left(\sum_{j=1}^{t} D_{ji}\right)\right]$$

$$: t \left[\emptyset + \sum_{i=1}^{K} \sum_{j=1}^{t-1} h_i (j-1) D_{ji} - \sum_{i=1}^{K} \left(\sum_{j=1}^{t-1} D_{ji}\right) G_i\left(\sum_{j=1}^{t-1} D_{ji}\right)\right]$$

then we have,

$$(t-1) \left[\sum_{i=1}^{K} h_i (t-1) D_{ti} - \sum_{i=1}^{K} \left(\sum_{j=1}^{t} D_{ji}\right) G_i\left(\sum_{j=1}^{t} D_{ji}\right)\right]$$

$$: \emptyset + \sum_{i=1}^{K} \sum_{j=1}^{t-1} h_i (j-1) D_{ji} - t\left[\sum_{i=1}^{K} \left(\sum_{j=1}^{t-1} D_{ji}\right) G_i\left(\sum_{j=1}^{t-1} D_{ji}\right)\right]$$

$$(t-1)^2 \sum_{i=1}^{K} h_i D_{ti} - (t-1) \sum_{i=1}^{K} \left(\sum_{j=1}^{t} D_{ji}\right) G_i\left(\sum_{j=1}^{t} D_{ji}\right)$$

$$: \emptyset + \sum_{i=1}^{K} \sum_{j=1}^{t-1} h_i (j-1) D_{ji} - t \sum_{i=1}^{K} \left(\sum_{j=1}^{t-1} D_{ji}\right) G_i\left(\sum_{j=1}^{t-1} D_{ji}\right)$$

This search will test if the left hand item is greater than the right hand item, indicating that there is a local minimum total cost per period. The above expression is somewhat complicated. Let's consider a few special cases here.

If $G_i\left(\sum_{j=1}^{t} D_{ji}\right) = G_i\left(\sum_{j=1}^{t-1} D_{ji}\right) = G_i'$

that happens when both quantities $\sum_{j=1}^{t} D_{ji}$ and $\sum_{j=1}^{t-1} D_{ji}$ fall in the same discount bracket. Then both will have the same discount rate, but the above comparison will become:

$$(t-1)^2 \sum_{i=1}^{K} h_i D_{ti} - (t-1) \sum_{i=1}^{K} D_{ti} G_i'$$

$$: \emptyset + \sum_{i=1}^{K} \sum_{j=1}^{t-1} h_i (j-1) D_{ji} - \sum_{i=1}^{K} \sum_{j=1}^{t-1} D_{ji} G'_i$$

which can be simplified into:

$$\sum_{i=1}^{K} (t-1) h_i D_{ti} - \sum_{i=1}^{K} D_{ti} G'_i$$

$$\frac{: \emptyset + \sum_{i=1}^{K} \sum_{j=1}^{t-1} h_i (j-1) D_{ji} - \sum_{i=1}^{K} \sum_{j=1}^{t-1} D_{ji} G'_i}{t-1} = R_o$$

This conclusion is similar to the one in Theorem 4-4. Only now the discount is put into consideration; therefore, the holding cost of the current period is modified into the holding cost subtracting the discount saving of the current period. R_0 thus obtained will take discount as a decision factor.

In the discount situation, it may have several discount levels available, and each next discount level may offer a significant saving. Therefore, each time when a local minimum total cost per period is found, it is worthwhile to extend the search to see if an additional quantity in the order will bring a better deal. To extend the ordering quantity $\sum_{j=1}^{t-1} D_{ji}$ of i^{th} items to $\sum_{j=1}^{t+k} D_{ji}$ may bring the i^{th} item into a new discount rate, but at the same time those additional quantities will cost extra holding costs. Once again, a comparison is needed to make the decision whether such extension to get the discount advantage is desirable. The comparison will be:

$$\sum_{j=1}^{t+k} h_i (j-1) D_{ji} : \sum_{j=1}^{t+k} D_{ji}\, G_i\Big(\sum_{j=1}^{t+k} D_{ji}\Big) - \sum_{j=1}^{t-1} D_{ji}\, G_i\Big(\sum_{j=1}^{t-1} D_{ji}\Big).$$

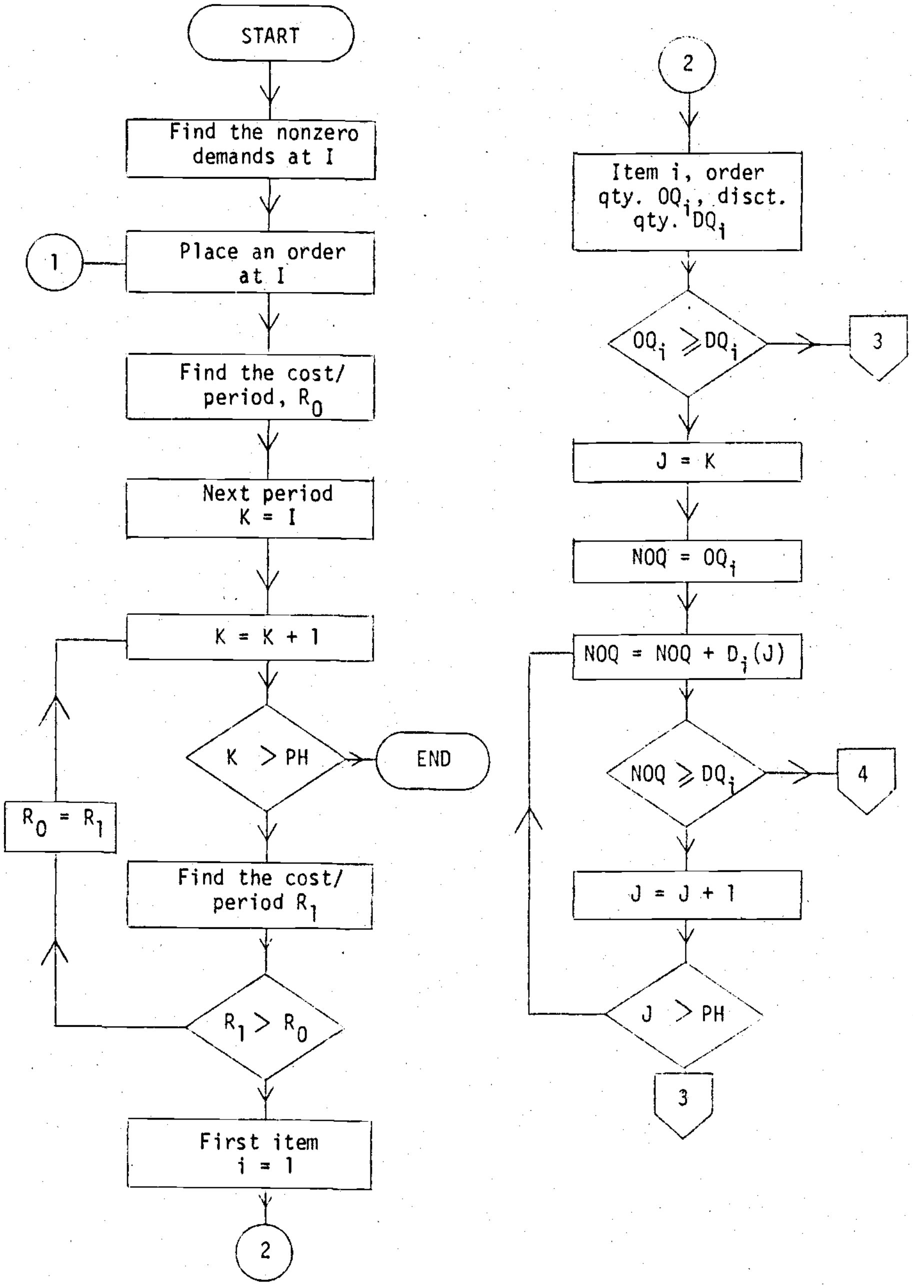
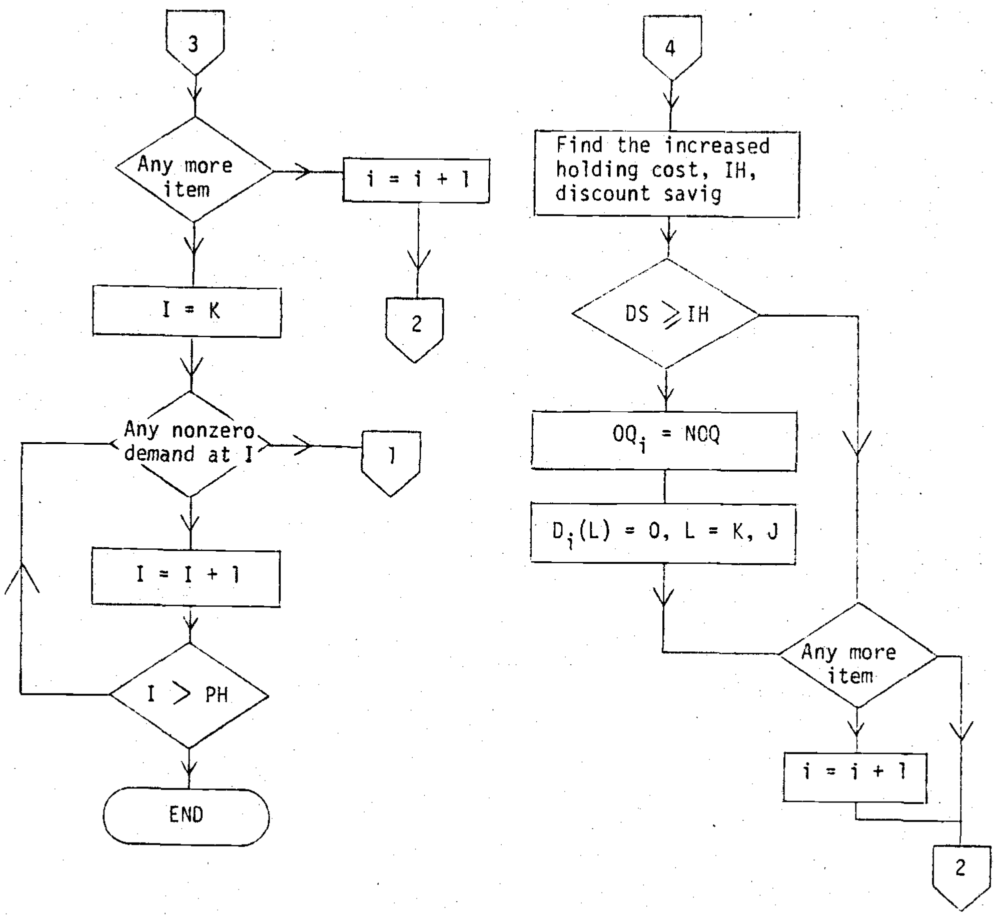


Figure 4-3 Flowchart of the Program-2



Flowchart of the Program-2 (Continued)

The check will repeat if there are more than one discount rate available.

In a brief summary, the program places the first order at the period where the first non-zero demand is found. The order will satisfy the demands of that period and the demands of the subsequent periods until a local minimum cost per period is found. In order to obtain some potential discount savings, the order may be extended to cover the demands of further periods depending on the trade-off between the increased holding costs and the discount advantages. The program will then place the next order at the period where the subsequent unfulfilled demand is found. Such procedures will be repeated until the end of the planning horizon is reached.

Heuristic Program 3

No Discount Situation

The traditional Wilson's Economic Ordering Quantity (EOQ) assumes that the optimal solution reaches when the ordering cost equals to the holding cost. This criterion works quite unstaisfactorily in a dynamic lot sizing problem. Yet there are a number of dynamic lot sizing methods derived from this criterion. This heuristic program, basically speaking is one of those methods. For each period, it decides whether or not to place an order based upon the comparison of the ordering cost and the holding cost. But since there are multiple items involved, considerations during each step must apply to everyone of them.

Since no backlog is allowed when there is no discount, we will consider either ordering all demands of a period in an order, or leave all of them to the next order. So the first step is to search the period with some non-zero demands and to place the order. This step is just like the other two algorithms that have already been derived. Then to decide whether the subsequent period should be covered by that order is determined by the comparison of the period's holding cost to the ordering cost:

$$\sum_{i=1}^{K} h_i (t-L) D_{ti} \geq 0$$

L denotes the period the order is placed. If the holding cost is greater than the ordering cost, it shows a definite advantage in placing a new order rather to include that period's demand in the prior order in a no-discount case. But even within the periods that each has its holding cost less than the ordering cost, to insert an order within those periods may cause the subsequent total saving to be greater than the ordering cost:

$$\sum_{j=j_1}^{j_2} \sum_{i=1}^{K} h_i (j_1 - L) D_{ji} \geq 0$$

where j_1 represents the period that a new order is placed, and j_1 to j_2 represent the periods being affected by the inserting new order.

Both considerations are reasonings that are true regardless of whether the distribution of demands is continuous or discrete They simply compare placing an order against not placing an order, and see which one costs less. From the comparison of different approaches in Chapter 3, we know the success of this approach over the others. To distinguish the solution obtained this way from the improved solution in backtracking, the above steps are referred to as the Phase-I method.

In some cases, the solutions from the Phase-I approach can be improved. During the Phase-I search, each order is placed, and we perform search in its range; the search, therefore, is unaffected by the planning that occurs before the ordering period. Suppose that the prior order covers a long range, the last few periods may carry some heavy holding costs, while the current order covers small quantity of demands and short range in coverage. We may thus find that moving the current order ahead of a period, or some periods, may reduce the total holding cost.

Let's suppose that L_1, L_2 and L_3 are the three consecutive ordering periods found from the Phase-I search. By moving the ordering period from L_2 to L_2', the reduced holding cost will be:

$$\sum_{j=L_2'}^{L_2-1} \sum_{i=1}^{K} (L_2' - L_1) \, h_i D_{ji}$$

while the increased holding cost will be:

$$\sum_{j=L_2}^{L_3-1} \sum_{i=1}^{K} (L_2 - L_2') \, h_i D_{ji}$$

The comparison of two items will tell whether such a move is desirable. This backtrack search is named the Phase-II search. Again the procedures in this phase are indifferent to the demand's characteristics. Phase-II will only improve the Phase-I solution, and guarantees no worse final solution than the Phase-I's result.

Discount Situation

In a discount situation, the possible discount saving from ordering certain amount of quantity shall be put into consideration. For a multiple-item dynamic lot sizing problem, the discount situation is complicated when the ordering cost is a joint one. The difficulties arise from the following factors:

1. Different items will have different discount rates for the different required quantities.
2. The dynamic lot requirements for each item may have a significant difference within an ordering period.
3. In order to get discount saving, one order may cover different item's requirements up to different periods.
4. The program-3 moves the tentative next order to and fro from the periods in order to search for a better solution involving the multiple-item and will cause inconvenience during such search.

The procedures of the program-3 will be as follows. Start at period with some non-zero demands, and place the first order to avoid backlog. Begin with the next period, the model will first be treated as an aggregated model, and be tested if the aggregated holding cost is greater than the ordering cost, i.e.

$$\sum_{i=1}^{K} h_i(j-L)D_{ji} \geq 0$$

L is the period the order being placed, and j is the tested period. When such period is found, the period will be set as the one to place the tentative next order. The next step is to see if an order is inserted between the current order and the tentative next order would lead to any saving. If so, the saving will come from the decreased holding cost of all items involved minus the new ordering cost:

$$\sum_{j=L}^{L_n-1} \sum_{i=1}^{K} h_i(L' - L)D_{ji} - 0$$

where L' is the period to insert the new order, L_n is the period to place the next order. Once the period to place the next order is tentatively decided, that which implies the range the current order will cover is found, the next consideration is whether the availability of discounts will lead to any saving.

Each item may have different discount quantity requirements, and may even have different numbers of discount levels available. The check is made to see if the quantity in the current order has already exceeded the highest discount level. If not, this implies the possibility to get some more saving from ordering extra quantities. The trade-off will come from the discount saving against the increased holding cost. For item m, suppose the quantity in the current order is $\sum_{j=L}^{L_n-1} D_{jm}$, where

L_n is the period to place the tentatively setting next order. In order to get the next discount level, the quantity will have to increase to:

$$\sum_{j=L}^{L_1} D_{jm} = \sum_{j=L}^{L_n-1} D_{jm} + \sum_{j=L_n}^{L_1} D_{jm} = D_1 + D_2$$

the increased holding cost is

$$\sum_{j=L_n}^{L_1} D_{jm}(L_n - L)h_m$$

and the discount saving depends on the increased quantity. Suppose the current discount level is $G(D_1) = G_1$, and after increasing the quantity D_2, the discount level changes to $G(D_1 + D_2) = G_2$, the discount saving from D_1 is:

$$D_1(G_2 - G_1)$$

while the discount on D_2 becomes:

$$D_2G_2 - D_2G(D_2).$$

Therefore, comparing

$$D_1(G_2 - G_1) + D_2G_2 - D_2G(D_2) : \sum_{j=L_n}^{L_1} D_{jm}(L_n - L)h_m$$

and the result will tell where such increment is desirable. Here an assumption is made that D_2 will have the discount rate $G(D_2)$ if D_2 is not included in the order. Of course, no guarantee can be made that this assumption will always hold true. Unless the ordering quantity of the next order is known, the discount rate applied to the demands in the D_2 will not be known.

The same procedures will be used to check for the next higher discount rate if it is available, and such procedure will also apply to all items involved in the order.

After all the items in the current order are checked, the procedure will search for the unfilled non-zero demand to place the next order, and in this way to plan for the whole horizon.

Since the planning of each order watches only the demanding situations after the ordering period, and regardless of what happens in the periods ahead, it will cause favorable changes if the order is moved ahead or combines with the previous order, and this does happen at a single item situation. But since multiple items are involved, here such backtrack check will lead to a complex situation. Because different item has different discount requirements, therefore when one order is moved ahead of a period(s), the consequence will not only change the holding costs of both orders that are involved, but also will possibly change the discount situations of different items in the two periods. To make this heuristic program simple, the only check made is to combine an order to the previous one, and see if the saving of an ordering cost, and increased discount saving from the later order is greater than the increased holding cost from the later order. This backtrack check for improvement will be repeated until all tentatively set orders (except the first one) are checked. This completes the procedure of Program-3 to search for the solution.

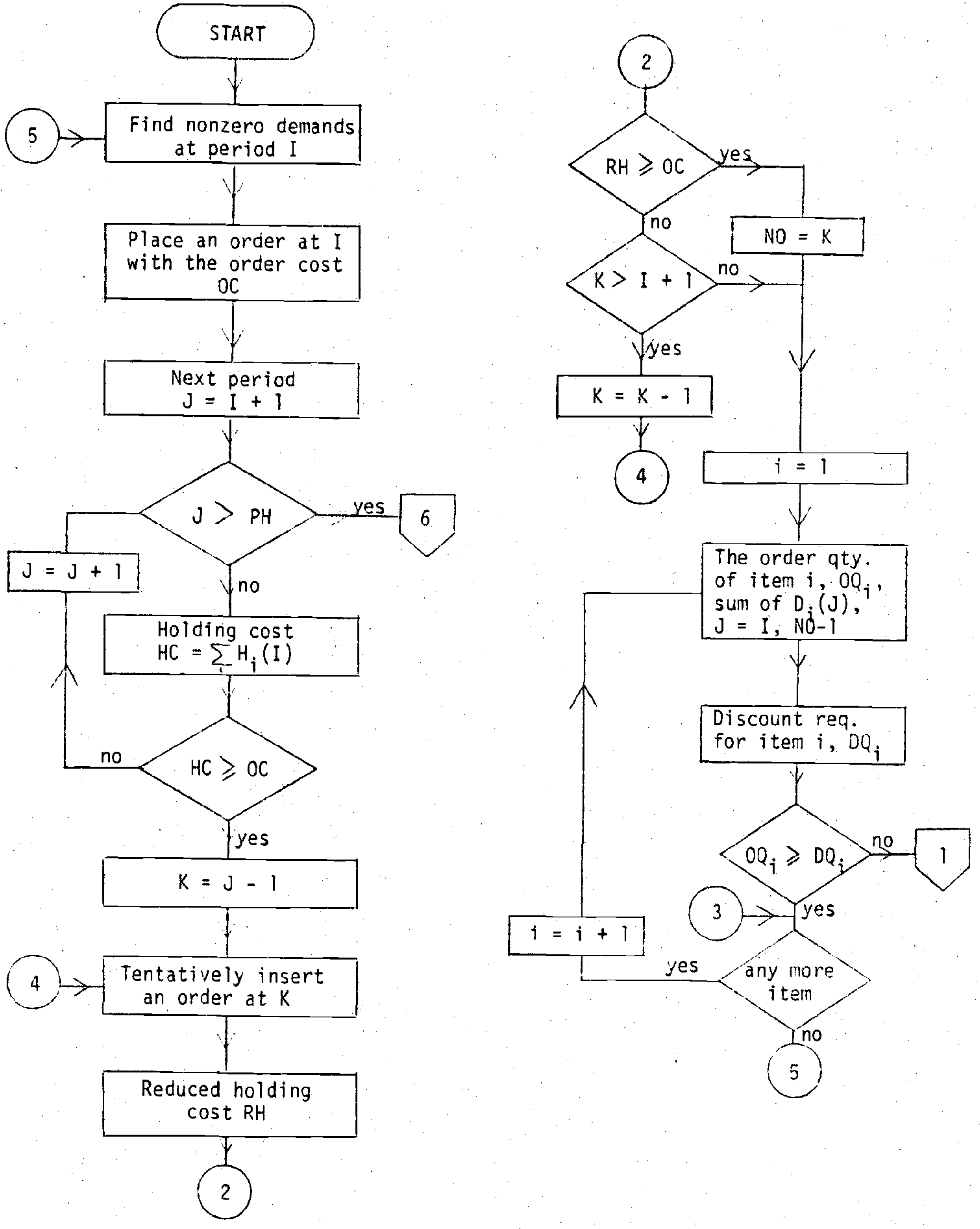
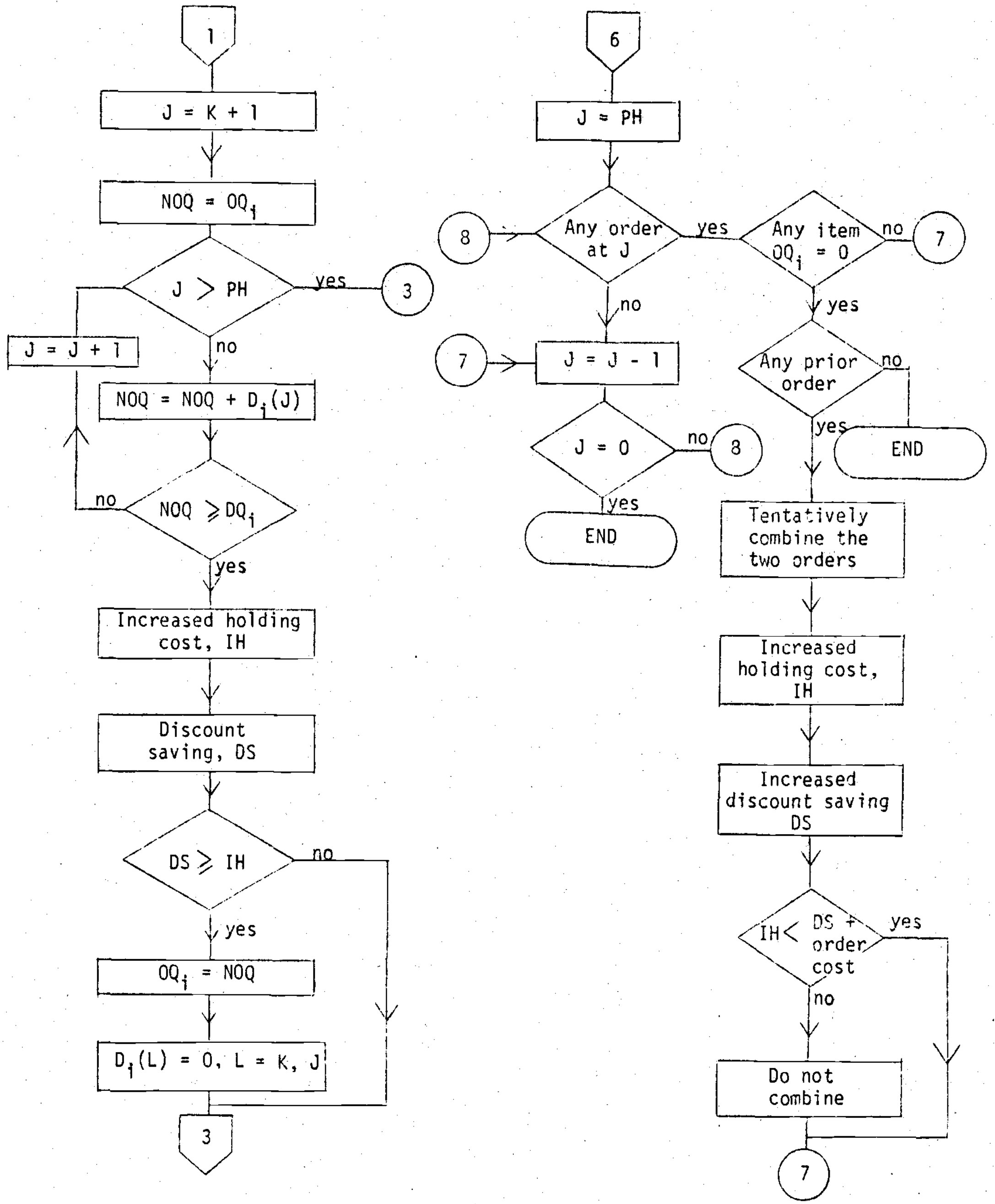

START
Find nonzero demands at period I
5
Place an order at I with the order cost OC
Next period J = I + 1
J > PH
yes
6
no
J = J + 1
Holding cost HC = $\sum_i H_i(I)$
HC ≥ OC
no
yes
K = J - 1
4
Tentatively insert an order at K
Reduced holding cost RH
2
2
RH ≥ OC
yes
no
NO = K
K > I + 1
no
yes
K = K - 1
4
i = 1
The order qty. of item i, OQ_i, sum of $D_i(J)$, J = I, NO-1
Discount req. for item i, DQ_i
$OQ_i \geq DQ_i$
no
1
3
yes
i = i + 1
yes
any more item
no
5

Figure 4-4 Flowchart of the Program-3



Flowchart of the Program-3 (Continued)

Development of Optimum Algorithm for the Joint Order Multiperiod Multiple-Item Dynamic Lot Sizing Problems when Discounts are Available

No Discount Situation

Dynamic Programming had been used to search for the optimum solution of a single-item dynamic lot sizing problem where the accumulated periods are treated as stages. The recursive equation representing the ordering and the holding cost is developed as:

$$V_j^* = \min_{m \leq j} (L_{mj} + V_{m-1}^*) \qquad V_{-1}^* = 0 \quad \text{where}$$

$$L_{mj} = H \sum_{L=m}^{j} D_L \,(L-m) + 0\, \delta_{mj}$$

$$\delta_{mj} = \begin{cases} 0 \text{ if } \sum_{L=m}^{j} D_L = 0 \\ 1 \text{ if } \sum_{L=m}^{j} D_L > 0 \end{cases}$$

The planning horizon theorem states that, in a forward algorithm:

$$V_n^* = L_{kn} + V_{k-1}^*$$

for planning up to the nth period, then when the planning horizon is extended to t>n

$$V_t^* = \min_{k \leq m \leq t} (L_{mt} + V_{m-1}^*), \; V_{-1}^* = 0$$

For a multiple-item problem, a similar expression can be developed. The optimal solution to plan up to the tth period will be:

$$(V_t^1 \, V_t^2 \, . \, . \, . \, V_t^k\,)^* = \min_{m_1 m_2 \ldots m_k \leq t} [(V_{m_1-1}^1 \, V_{m_2-1}^2 \, . \, . \, . \, V_{m_k-1}^k)^* + L_{m_1 t}^1 \, L_{m_2 t}^2 \, . \, . \, . \, L_{m_k t}^k]$$

where $(V^1_{m_1} V^2_{m_2} \dots V^k_{m_k})^*$ represents the optimal solution to plan up to m_1 *th* period for item 1, m_2 *th* period for item 2, and so on.

$$L^1_{m_1 t} L^2_{m_2 t} \dots L^k_{m_k t} = \sum_{i=1}^{k} \sum_{L=m_i}^{t} H_i D_{Li}(L-m_i) + \emptyset \, \delta_{m_1 m_2 \dots m_k t}$$

where

$$\delta_{m_1 m_2 \dots m_k} = \begin{cases} 0 \text{ if } \sum_{i=1}^{k} \sum_{L=m_i}^{t} D_{Li} = 0 \\ 1 \text{ if } \sum_{i=1}^{k} \sum_{L=m_i}^{t} D_{Li} > 0 \end{cases}$$

The above expression is the most general one. In practice, such general expression represents a very time-consuming search. While most of the combinations actually do not need to be considered, the following theorem will simplify such search procedures.

Theorem 4-5: In the forward planning of a joint-order multiple-item no discount problem, the multiple-item problem can be treated as an aggregate problem, that means if there is no discount available.

$$(V^1_t V^2_t \dots V^k_t)^* = \min_{m_1 m_2 \dots m_k < t} [(V^1_{m_1-1} V^2_{m_2-1} \dots V^k_{m_k-1})^* + L^1_{m_1 t} L^2_{m_2 t} \dots L^k_{m_k t}]$$

At optimal situation, the only combination needs to be considered is at $m_1 = m_2 = \dots = m_k$

Proof: Suppose not all m_i are equal. Let

$$m_n = \min_{g=1,2,\dots k} (m_g)$$

then

$$L^1_{m_1 t} \; L^2_{m_2 t} \dots L^k_{m_k t} = \sum_{i=1}^{k} \sum_{L=m_i}^{t} H_i D_{Li}(L - m_n) + \emptyset \, \delta_{m_1 m_2 \dots m_k t}$$

That means the order will be placed at the earliest period with some unfilled non-zero demands in order to avoid the backlog.

Assuming the immediate prior order to be placed is at h^{th} period, the holding cost from the priod m_n to t is

$$\sum_{i=1}^{k} f(m_i) + \sum_{i=1}^{k} \sum_{L=m_i}^{t} D_{Li}(L-m_n)H_i$$

where

$$f(m_i) = \begin{array}{ll} \displaystyle\sum_{L=m_n}^{m_i-1} H_i D_{Li}(L - h) & m_n < m_i \\ 0 & m_n = m_i \end{array}$$

and since $m_n > h$,

$$\sum_{L=m_n}^{m_i-1} H_i D_{Li}(L - h) > \sum_{L=m_n}^{m_i-1} H_i D_{Li}(L-m_n)$$

We note, therefore, that every order will take care all the requirements from the period to place an order up to the period immediately prior to the next order. Therefore, in the search of the optimum solution, only

$$(V^1_{m_n-1}\ V^2_{m_n-1}\ \ldots\ V^k_{m_n-1})^* + L^1_{m_n t}\ \ldots\ L^k_{m_n t} \qquad m_n = 1,\ldots,n$$

will be considered.

This indicates whenever an order is placed, the order will cover all items' demands, with no exception, up to the period prior to the next order. Thus the multiple-item problem can essentially be treated as

$$T_j = \sum_{i=1}^{k} D_{ji}$$

When calculating the holding cost, one must remember that each item may have different holding cost rate. If an order is placed at g^{th} period,

$$H = \sum_{i=1}^{k} H_i D_{ji} (i - g)$$

represents the holding cost generated from j^{th} period's demands.

Discount Situation

Once the discount is available, the problem becomes complicated. One order may cover different items' demands up to different periods. For each period, besides the decision in placing an order or not, the item's demand quantity included in the order also needs to be considered. The problem can no longer be treated as an aggregated problem. However, the dynamic programming method that works for other simpler situations can also be used for this complicated problem.

In the forward planning process, the recurssive form of optimum planning of t periods are as follows:

$$(V_t^1 V_t^2 \ldots V_t^k)^* = \min_{m_1, m_2 \ldots m_k \leq t} [(V_{m_1-1}^1 V_{m_2-1}^2 \ldots V_{m_k-1}^k)^* + L_{m_1 t}^1 L_{m_2 t}^2 \ldots L_{m_k t}^k]$$

and

$$L_{m_1 t}^1 L_{m_2 t}^2 \ldots L_{m_k t}^k = \sum_{i=1}^{k} \sum_{L=m_i}^{t} H_i D_{Li} (L - n) + \emptyset \, \delta_{m_1 m_2 \ldots m_k t} - \sum_{i=1}^{k} G(\sum_{L=m_i}^{t} D_{Li}) \sum_{L=m_i}^{t} D_{Li}$$

where

$$n = \min_{i=1,2,\ldots k} m_i$$ in order to avoid backlog.

$$\delta_{m_1 m_2 \ldots m_k t} = \begin{cases} 1 & \sum_{i=1}^{k} \sum_{L=m_i}^{t} D_{Li} > 0 \\ 0 & \sum_{i=1}^{k} \sum_{L=m_i}^{t} D_{Li} = 0 \end{cases}$$

It is easy to note that in following these kinds of approaches, a tremendous amount of combinations will be searched in order to find an optimum solution.

For example, in order to deal with a two-item single discount joint-order problem, an exhaustive search for a five-period planning horizon problem is needed to search for $4^4 = 256$ different outcomes in order to get the best solution. In general, for an n-period two-item single discount problem, each period may have four kinds of outcomes: (1) order for both items, (2) no order is placed, (3) order only for the first item, and (4) order only the second item. So the total number of possible outcomes is 4^{N-1}. When the first period with non-zero demand is designed to place an order, it only has one outcome.

The Planning Horizon Theorem, developed by Wagner-Whitin, has simplified the searching procedures to get an optimal solution. The extension of Planning Horizon Theorem for the discount situation, developed in Chapter II, brings the information to reduce the search in a discount situation if certain conditions can be fulfilled. Similarly, here the extension of Planning Horizon Theorem to a multiple-item discount problem is developed for the same purpose.

Theorem 4-6: In a forward algorithm, if the minimum cost decision of planning up to period t_n is through ordering quantity $\sum_{L=q_i}^{t_n} D_i \ (L)$ of i^{th} item at period $t_q = \min_{i=1,2,3\ldots k} q_i$, and $\sum_{L=q_i}^{t_n} D_{Li} \geq B_{ii_J}$, i=1,2,..., K, is the highest discount level for the i^{th} item, then in order to extend the planning horizon to t_m^{th} period, $t_m > t_n$, it is sufficient to consider only periods j, $t_q \leq j \leq t_m$ (in other words, the combinations involving $\sum_{L=j}^{t_n} D_{Li}$, $j < t_q$ need not be considered).

Proof: Let C_Q represent the total cost of planning up to the period t_n through ordering $\sum_{L=q_i}^{t_n} D_{Li}$, i=1,2,...k at the priod $t_q = \min_{i=1,2,\ldots k} q_i$. When the planning horizon extends to the t_m^{th} period, $t_m > t_n$, the cost through ordering $\sum_{L=q_i}^{t_m} D_{Li}$, i=1,2,...k is:

$$C_Q' = C_Q + \sum_{i=1}^{k} \sum_{L=t_n+1}^{t_m} D_{Li} h_i (L - t_q)$$

$$- \sum_{i=1}^{K} [G(B_{ii_J}) \sum_{L=t_n+1}^{t_m} D_{Li}]$$

and the total cost through ordering $\sum_{L=j_i}^{t_n} D_{Li}$, i=1,2,...k at period $j = \min_{i=1,2,\ldots k} j_i$ and $j < t_q$ is C_J. When the planning horizon extends to the t_m^{th} period, $t_m > t_n$, the cost through ordering $\sum_{L=ji}^{t_m} D_{Li}$, i=1,2,...,k is:

$$C_J' = C_J + \sum_{i=1}^{k} \sum_{L=t_n+1}^{t_m} D_{Li} h_i (L - j)$$

$$- \sum_{i=1}^{k} [G(B_{ii_J}) \sum_{L=t_n+1}^{t_m} D_{Li}]$$

therefore, $C_J \geq C_Q$, $(L - j) > (L - t_q)$

which concludes $C_J' > C_Q'$. Thus to plan up to and period $t > t_n$, it is sufficient to consider only periods j, $t_q \leq j \leq t$.

Notice the sufficient condition is $\sum_{L=q_i}^{t_n} D_{Li} \geq B_{ii_J}$ for i=1,2,...k. The reason that every item has to be considered is because this is a joint-order multiple-item problem. If there is any item that cannot fulfill this condition, i.e. $\sum_{L=q_i}^{t_n} D_{Li} < B_{ii_J}$ for some i, then when the planning horizon extends to the period $j > t_k$, the increased discount rate from $G(\sum_{L=j_i}^{t_n} D_{Li})$ to $G(\sum_{L=j_i}^{t_m} D_{Li})$ may be greater than the increased discount rate from $G(\sum_{L=q_i}^{t_n} D_{Li})$ to $G(\sum_{L=q_i}^{t_m} D_{Li})$. If the difference of the two increased discount rates causes higher saving than the difference of the two additional holding costs, it is possible to find that placing the order at j is a better deal.

CHAPTER V

SITUATION WHEN SPLIT ORDERS ARE ALLOWED

Solutions with Split Orders

Almost all the dynamic lot sizing techniques that developed assume the whole lot situation. That is, the solution is formed by the whole lots (demands) only, and no split order is allowed. When the discounts are available, in order to get the cost reduction from the discounts, sometimes it is worthwhile to order some extra quantity to reach the minimum quantity requirements to get the discount. But the extra demand put into the order may be larger than it is desired. Only a portion of that demand may be enough to qualify the order for the discount. A split lot in such situations may be appropriate to give a better answer.

Network Model

Such problem can be described through the network flowchart. The flowchart contains several symbols. A square stands for the inventory decision, either to fulfill the current period's demand, or to store for future usage. A circle indicates the ordering quantity, or demands. Since each item may have several discount levels, a special symbol is used to indicate the "exclusive or" decision in choosing the order's quantity for one of the discount levels, including not to order at all. The character of "exclusive

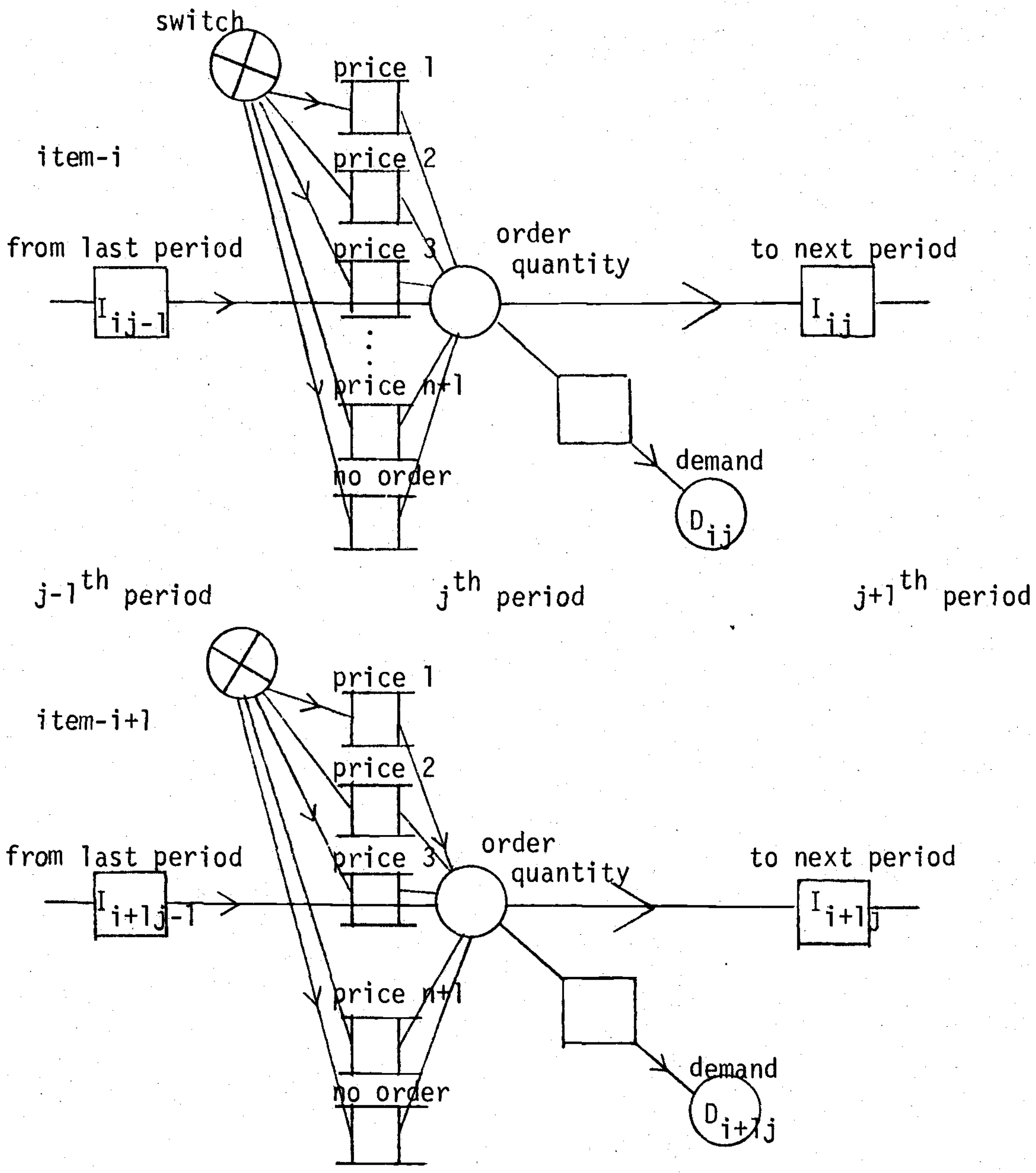


Figure 5-1 Network of a Multiperiod Multiple-Item Dynamic Lot Sizing Problem When Discounts Are Available

or" allows the value 0 or 1, so this special symbol is used also to indicate that such solution value must be an integer. A part of the network of a multiperiod multiple-item dynamic lot sizing problem with discounts available is shown at Figure 5-1.

Mathematical Programming Model

The optimal model can be formed by the mixed integer programming model. The objective of the model is to minimize the sum of the ordering cost, holding cost and the purchasing costs, and the purchasing unit costs are determined by the order quantities. A general model for the multiperiod multiple-item multiple discount-level dynamic lot sizing problem is developed as the following:

D_{ki} : Demands for the k^{th} item at the i^{th} period.

$\emptyset$: Ordering cost.

H_k : Holding cost per unit per period for the k^{th} item.

P_{kj} : Purchasing cost for the k^{th} item at the j^{th} price.

B_{kj} : The minimum purchasing quantity to get the j^{th} discount for the k^{th} item.

I_{ki} : The ending inventory for the k^{th} item at the i^{th} period.

δ_{kij} : When it equals 1, it represents to place order for the k^{th} item at the i^{th} period at the j^{th} price, otherwise it will equal to zero.

δ_{kiw} : When it equals 1, it represents no order is placed for the k^{th} item at the i^{th} period.

X_{kij} : Order quantity for the k^{th} item at the i^{th} period and j^{th} price.

The model can be formed:

$$\text{MIN} \sum_{k=1}^{K} \sum_{i=1}^{N} H_k I_{ki} + \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=0}^{J_k} P_{kj} X_{kij} + \sum_{i=1}^{N} (1 - \prod_{k=1}^{K} \delta_{kiw}) \emptyset$$

subject to:

$$X_{ki0} - (B_{k1} - 1) \delta_{ki0} \leq 0$$

$$X_{kij} - B_{kj}\delta_{kij} \geq 0$$

$$X_{kij} - (B_{kj+1} - 1) \delta_{kij} \leq 0$$

$$X_{kiJ_k} - (\sum_{i=1}^{N} D_{ki}) \delta_{kiJ_k} \leq 0$$

$$\sum_{j=0}^{J_k} X_{kij} + I_{ki-1} - D_{ki} = I_{ki}$$

$$\sum_{j=0}^{J_k} \delta_{kij} + \delta_{kiw} = 1$$

$k = 1, 2, \ldots, K$

$i = 1, 2, \ldots, N$

$j = 1, 2, \ldots, J_k$

$\delta_{kij} = 0$ or 1

The factor $\sum_{i=1}^{N} (1 - \prod_{k=1}^{K} \delta_{kiw}) \emptyset$ in the objective function makes the model a geometrical mixed integer programming model. Due to the special character of our planning situation, we know that

$$\prod_{k=1}^{K} \delta_{kiw} = \begin{cases} 1 \text{ if } \delta_{1iw} = \delta_{2iw} = \cdots = \delta_{Kiw} = 1 \\ 0 \text{ otherwise} \end{cases}$$

To take advantage of this character, we can simplify the model by defining the following additional variables:

$$DT_i = \prod_{k=1}^{K} \delta_{kiw} \qquad i = 1, 2, \ldots, N$$

and increase the following constraints:

$$DT_i \leq \prod_{k=1}^{K} \delta_{kiw} / K \qquad i = 1, 2, \ldots, N$$

The following linear mixed integer model will effectively work out the same information as it comes from the previous geometrical mixed integer model:

$$\text{MIN} \sum_{k=1}^{K} \sum_{i=1}^{N} H_k I_{ki} + \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=0}^{J_k} P_{kj} X_{kij} + \sum_{i=1}^{N} (1 - DT_i) \emptyset$$

subject to:

$$X_{ki0} - (B_{k1} - 1) \delta_{ki0} \leqslant 0$$

$$X_{kij} - B_{kj}\delta_{kij} > 0$$

$$X_{kij} - (B_{kj+1} - 1) \delta_{kij} \leqslant 0$$

$$X_{kiJ_k} - (\sum_{i=1}^{N} D_{ki}) \delta_{kiJ_k} \leqslant 0$$

$$\sum_{j=0}^{J_k} X_{kij} + I_{ki-1} - D_{ki} = I_{ki}$$

$$DT_i \leqslant (\sum_{k=1}^{K} \delta_{kiw}) / K$$

$$\sum_{j=o}^{J_K} \delta_{kij} + \delta_{kiw} = 1 \qquad k = 1, 2, \ldots, K$$

$$i = 1, 2, \ldots, N$$

$$\delta_{kij}, \delta_{kiw}, DT_i = 0 \text{ or } 1 \qquad j = 1, 2, \ldots, J_k$$

Discussion of the Problems to Search the Solution

Although the model can be set up theoretically, the practicality is still in question. Two approaches have been attempted to search for an optimum solution from the model. The first one is Gomory's All Integer algorithm. The approach is both time and cost consuming.

A two-item single discount level problem was used as an example. It took over a hundred iterations and $50.00 computer time on Cyber 73, while the solution is still far away from an acceptable answer. The second approach is Branch-and-Bound Mixed Integer Algorithm. Unfortunately, this approach requires a huge extended core memory to store the intermediate results. A table is developed for the two-item problems (Table 5-1). Even a five-period two-item single discount level problem will require more than 400,000 octal core memory space, while the maximum core memory space available at OSU's Cyber 73 is 144,000 octal space. Therefore, practically speaking, the mixed integer programming model, although it guarantees an optimal solution, has a limited application value for a large size problem such as a general multiperiod multiple-item multiple-discount dynamic lot sizing problem. Table 5-1 lists some information about the mathematical programming model of a two-item dynamic lot sizing problem. Information includes the number of variables, the number of integers, and the number of constraints in the mode, and the required core memory size to solve such mathematical programming problems using the Branch-and-Bound approach. Information about three situations: (1) both items with no discount, (2) each of two items with one discount available, and (3) each of two items with two discounts available, are listed in the Table 5-1. The required core memory sizes using the Branch-and-Bound approach to solve a two-item dynamic lot sizing problem with the different period lengths and with the different number of discounts are plotted at Figure 5-2 for comparison.

Discount Situation	Information about the Math. Model	Number of Periods in the Planning Horizon						
		5	6	7	8	9	10	11
No Discount	No. of Variables	45	54	63	72	81	90	99
	No. of Integers	25	30	35	40	45	50	55
	No. of Constraints	35	42	49	56	63	70	77
	Required Core Memory Size (oct.)	130726	225726	351740	530526	745632	1225016	1552024
Each With One Discount	No. of Variables	65	78	91	104	117	130	143
	No. of Integers	35	42	49	56	63	70	77
	No. of Constraints	55	66	77	88	99	110	121
	Required Core Memory Size (oct.)	415306	711722	1316540	2047346	2737732	4003662	5236744
Each With Two Discount	No. of Variables	85	102	119	136	153	170	187
	No. of Integers	45	54	63	72	81	90	99
	No. of Constraints	75	90	105	120	135	150	165
	Required Core Memory Size (oct.)	1132766	2005116	3123740	4542166	6512732	11050726	14027064

Table 5-1. Information Table of a Two-Item Dynamic Lot Sizing Mathematical Programming Model

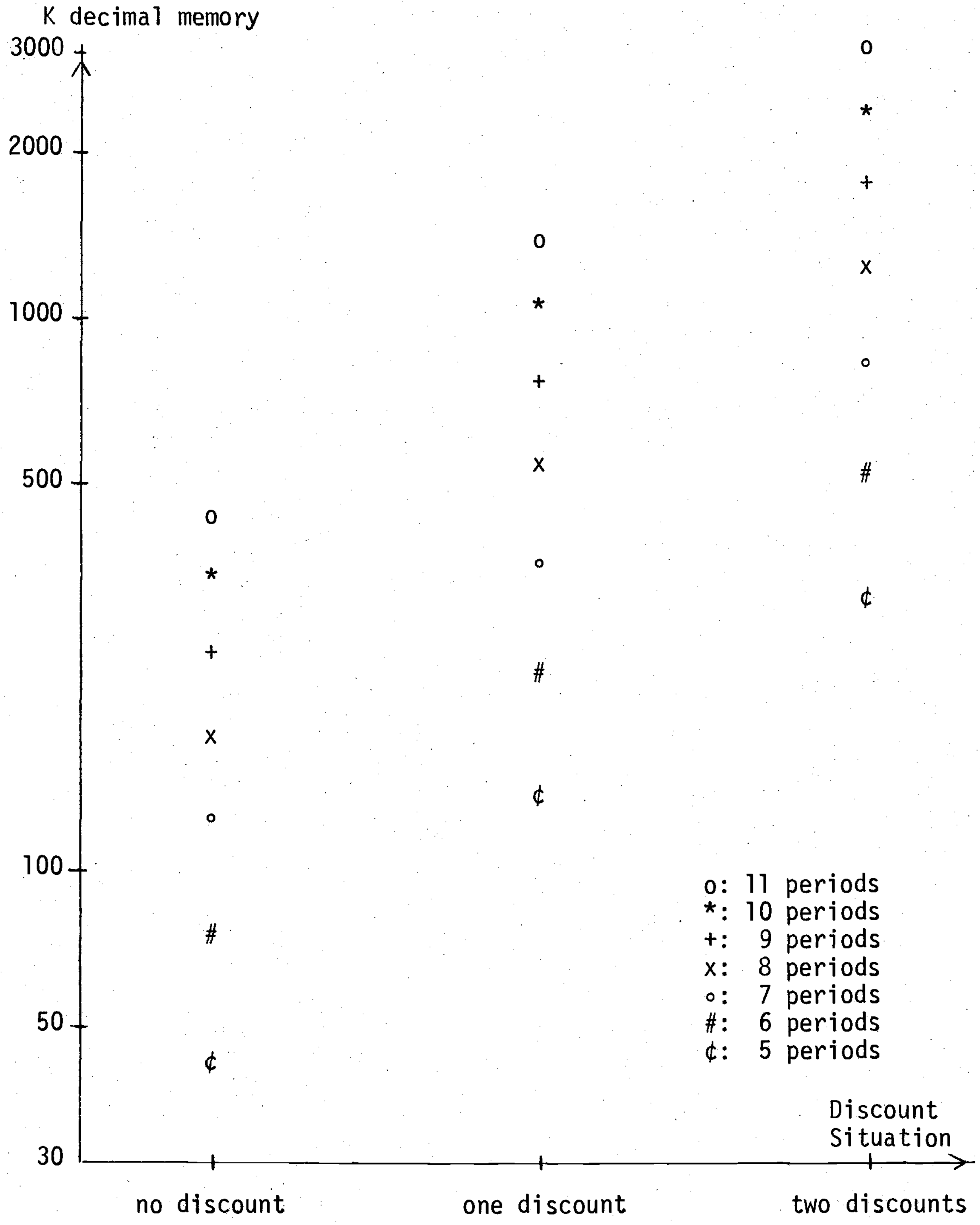


Figure 5-2 Required Core Memory Size to Search for an Optimum Solution Using the Branch-and-Bound Approach

CHAPTER VI

SOME NUMERICAL EXAMPLES OF TWO-ITEM PROBLEMS

Two-Item Problems

The simplest multiple-item problem is a two-item problem. For a dynamic, n-period, two-item, no-discount lot sizing model with non-zero demands at the first period (which means an order must be placed at the first period to avoid the backlog), the total number of feasible solution is 2^{n-1}. This comes from the fact that at each period, after the first, the decision-maker has the choice of either placing an order or not placing any. Under such situations, a two-item problem will appear as a basic aggregate problem. It can generally be extended to involve any number of items under the assumption that there is no discount available to any item. A characteristic of such problem is (Wagner, Whitin, 1958):

$$I_{t-1}X_t = 0$$

where

$$X_t = \begin{cases} 1 \text{ means placing an order at the } t^{th} \text{ period} \\ 0 \text{ means not placing an order at the } t^{th} \text{ period} \end{cases}$$

I_t represents the inventory at the end of the t^{th} period

$I_{-1} = 0$

Therefore, whenever an order is placed in the t^{th} period, the ending inventory of $t-1^{th}$ period must be zero. In other words, the prior order will not carry any inventory for the t^{th} demand if an order is to be placed during the t^{th} period.

Once there is any discount available to the items, the above characteristic will no longer exist. Except for the first period, the possible outcomes of each period may be (1) to place an order for both items, (2) to place an order for the demand of one of them while another item's demand was ordered in the prior order to get some discount advantages, or (3) to place no order. Therefore, for an n-period two-item dynamic lot sizing model, a problem under discount situation will have up to 4^{N-1} solutions. If it is a five-period problem, there will be 256 solutions, and if it is a 12-period problem, like most of the testing problems in this thesis, there will be 4,194,304 solutions. For a general K-item discount problem, there will be

$$\sum_{i=0}^{K} \binom{K}{i}^{N-1}$$

number of solutions. To search for an optimal solution among millions of solutions requires a special technique. While the dynamic programming approach in searching the optimal solution of a dynamic lot sizing problem is basically an exhaustive search, a large number of solutions must be investigated even when the planning horizon theorem is used to reduce the scope of the search. While the dynamic programming approach can promise an optimum solution, and serves as a good benchmark, its practicality is challenged by the tremendous computational cost of such tests. The CPU time required to reach an optimum solution in a multiperiod two-item discount problem has been measured and estimated (Figure 6-1). It is found that the required CPU time grows exponentially

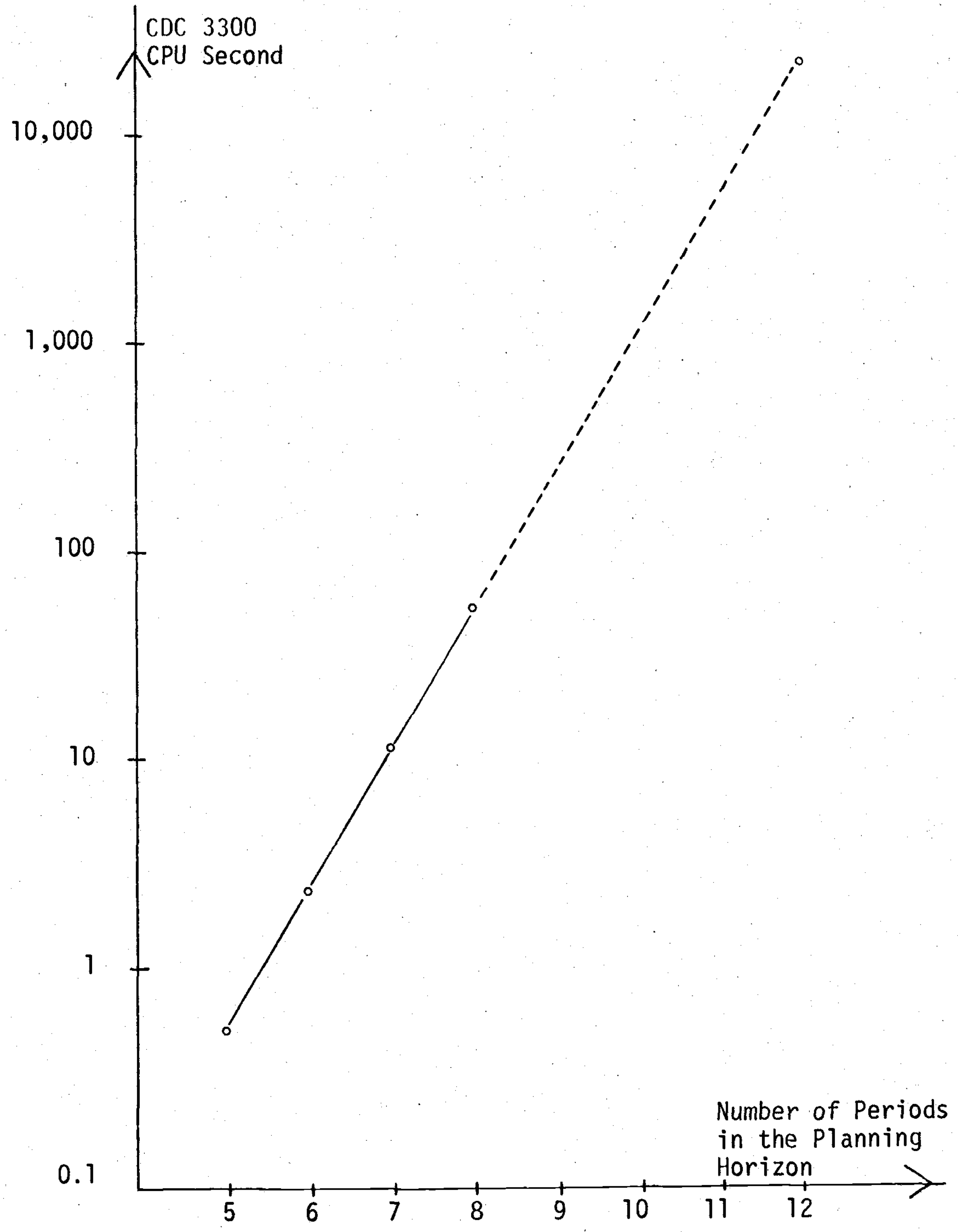


Figure 6-1 Required CPU Time to Search for an Optimum Solution for a Multiperiod Two-Item Single Discount Problem

as the number of periods increases. The estimated CPU time to reach an optimum solution in a 12-period problem is more than 16,000 sec.. In order to avoid such costly tests, the following limitations were imposed upon our experiments:

1. Dynamic programming approach was used to solve only the following benchmark problems:
   (i) when the ordering cost is equal to zero
   (ii) when there is no discount available.
2. Comparison tests that are too expensive to be solved by the dynamic programming approach were used only to compare heuristic programs among themselves.

When the ordering cost was very small, a two-item model was approximated by two single-item models, and when there was no discount available, the two-item model was approximated by an aggregate model. These approximations helped reduce the computational costs significantly, and made the dynamic programming approach feasible.

Test Data

The 100 sets of data used to test the performance of single-item dynamic lot sizing techniques were expanded to form the testing data for the performance tests of two-item dynamic lot sizing techniques. Two groups of data were assembled. The first group simply picked every two consecutive sets from the 100 sets of data, where the data were sorted in an ascending order according to the coefficients of variation, as a single set of two-item demands. A total of 50 sets of two-item

(1) Ten Sets of 12-Period Demands from *LDATA1:

105	134	134	75	105	105	60	105	90	90	30	72	.324
102	170	85	68	102	102	51	51	119	119	85	51	.382
96	112	32	112	48	48	32	96	144	144	112	129	.453
49	163	98	114	130	114	16	130	98	49	81	63	.457
91	61	61	15	91	151	106	121	151	106	121	30	.472
69	14	138	124	138	55	97	41	83	69	138	139	.473
119	60	75	60	134	45	134	134	15	60	149	120	.481
161	60	141	60	161	40	80	100	60	80	40	122	.482
123	105	105	158	35	140	123	18	53	53	123	69	.488
84	100	117	67	100	67	33	167	167	84	100	19	.489

(2) Ten Sets of 12-Period Demands from *LDATA4:

105	134	134	75	105	105	60	105	90	90	30	72	.324
178	0	107	36	0	125	0	143	178	36	143	159	.786
102	170	85	68	102	102	51	51	119	119	85	51	.382
121	40	40	100	121	201	201	181	40	40	0	20	.789
96	112	32	112	48	48	32	96	144	144	112	129	.453
181	20	60	20	0	60	201	100	20	121	121	201	.799
49	163	98	114	130	114	16	130	98	49	81	63	.457
181	0	201	80	181	60	0	0	80	40	161	121	.821
91	61	61	15	91	151	106	121	151	106	121	30	.472
42	0	104	167	21	42	208	0	42	146	167	166	.821

Table 6-1. Some Testing Data Sets

demands were formed in the first group. The group was named *LDATA1. The second group combined every set of data with its next 50th set from the original 100 sets of data to form a new set of two-item demands. Fifty other sets of two-item demands were formed in the second group of data. This group was named *LDATA4. The first five sets of data and their coefficients of variation from each group are listed in Table 6-1.

Selected Situations

No Ordering Cost

The first selected situation to be tested is when there is no ordering cost. The constraint that makes the joint-order multiple-item problems different from the single-item is the joint-ordering cost. When this cost disappears, a multiple-item problem will simply become a number of single-item problems. Each item will search for its own lot sizing decision regardless of the decisions from other items. Under such conditions, a joint-order two-item problem can be treated as two separate single-item problems. The optimum solution of that two-item problem can be found by repeatedly using the dynamic programming approach to a single-item model. The resulting optimum solution will be used as the benchmark to test the performance of the multiple-item heuristic programs.

Four different discount rates are used in the performance tests. The holding costs and the required units for discount are set at $2.00/unit/period, and 200 units for both items. Data from *LDATA1 and *LDATA4 are used as the testing data. The average results from every

Ordering Cost: $0

Holding Cost: both $2/unit/period

Qty. Req. for Discount: both 200 units

A: Testing with 50 Sets of Data: *LDATA1

Discount Rate	PRG-1	PRG-2	PRG-3	WW
$1	-636.30	-522.46	-587.54	-738.06
$2	-1382.20	-1766.68	-1935.64	-2295.24
$3	-2808.08	-3273.30	-3825.62	-4142.78
$4	-4469.44	-5093.68	-5712.72	-6073.36

B: Testing with Another 50 Sets of Data: *LDATA4

Discount Rate	PRG-1	PRG-2	PRG-3	WW
$1	-609.16	-505.34	-587.54	-738.06
$2	-1419.32	-1787.32	-1935.64	-2295.24
$3	-2581.38	-3387.52	-3825.62	-4142.78
$4	-3881.88	-5224.68	-5712.72	-6073.36

Table 6-2. Comparison of Average Costs from the Performance Tests When There Is No Ordering Cost

group of 50 sets of solutions are recorded in Table 6-2.

No Discount

The second selected situation to be tested is when there is no discount available. In such a situation, as explained at the beginning of this chapter, the demands of all the items in the same period must be satisfied simultaneously. If the decision is to place an order during that period, the order will cover the demands of all the items in that period. If the decision is not to place an order in that period, the demands of all the items in that period will be covered by the prior order. A no-discount problem has the characteristic:

$$I_{t-1}X_t = 0$$

where

$$X_t = \begin{cases} 1 \text{ when the decision is to place an order at } t^{th} \text{ period} \\ 0 \text{ when the decision is not to place an order at } t^{th} \text{ period} \end{cases}$$

I_t represents the ending inventory at t^{th} period

So if I_{t-1} is not equal to zero, which means that there is some ending inventory during the $t\text{-}1^{th}$ period, X_t must be set to zero. This means that we allow no order in the t^{th} period. If the demands of some items in the t^{th} period are covered by a prior order, and the demands of other items of the t^{th} period are left uncovered, then the ending inventory at the $t\text{-}1^{th}$ period will not be zero. This forces X_t=0, and leads to a backlog because some demands in the t^{th} period will be uncovered by any order. Therefore, the demands of all items in the

Holding Cost: both $2/unit/period

Discount Rate: $0

Qty. Req. for Discount: both 200 units

A: Testing with 50 Sets of Data: *LDATA1

Ordering Cost	PRG-1	PRG-2	PRG-3	WW
$ 300	3648.36	3019.48	3015.08	2983.52
$ 206	2589.80	2161.00	2158.12	2147.48
$ 120	1564.52	1294.96	1294.96	1292.52
$ 92	1250.00	1000.94	1000.76	1000.28
$ 48	584.24	527.88	527.88	527.88

B: Testing with Another 50 Sets of Data: *LDATA4

Ordering Cost	PRG-1	PRG-2	PRG-3	WW
$ 300	3724.88	3082.60	3075.12	3050.08
$ 206	2828.36	2213.56	2212.36	2202.84
$ 120	1697.32	1342.92	1342.92	1342.08
$ 92	1273.12	1044.32	1043.76	1043.72
$ 48	656.04	553.08	553.08	553.08

Table 6-3. Comparison of Average Costs from the Performance Tests When There Is No Discount Available

Holding Cost: both $2/unit/period

Discount Rate: $0

Qty. Req. for Discount: both 200 units

A: Testing using *LDATA1:

Ordering Cost	PRG-1	PRG-2	PRG-3
$ 300	664.84	35.96	31.56
$ 206	442.32	13.52	10.64
$ 120	272.00	2.44	2.44
$ 92	249.72	0.66	0.48
$ 48	56.36	0.00	0.00

B: Testing using *LDATA4:

Ordering Cost	PRG-1	PRG-2	PRG-3
$ 300	674.80	32.52	25.04
$ 206	625.52	10.72	9.52
$ 120	355.24	0.84	0.84
$ 92	229.40	0.60	0.04
$ 48	102.96	0.00	0.00

Table 6-4. The Comparison of Average Costs Over the Optimum Solutions When There Is No Discount Available

Holding Cost: both $2/unit/period

Discount Rate: $0

Qty. Req. for Discount: both 200 units

A: Testing using *LDATA1:

Ordering Cost	PRG-1	PRG-2	PRG-3
$ 300	22.3%	1.2%	1.1%
$ 206	20.6%	0.6%	0.5%
$ 120	21.0%	0.2%	0.2%
$ 92	25.0%	0.07%	0.05%
$ 48	10.7%	0%	0%

B: Testing using *LDATA4:

Ordering Cost	PRG-1	PRG-2	PRG-3
$ 300	22.1%	1.1%	0.8%
$ 206	28.4%	0.5%	0.4%
$ 120	26.5%	0.06%	0.06%
$ 92	22.0%	0.05%	0.04%
$ 48	18.6%	0%	0%

Table 6-5. The Comparison of Average Percentage Costs Over the Optimum Solutions When There Is No Discount Available

same period must be satisfied at the same time. This means, therefore, that a multiple-item problem can be treated as an aggregate problem, and single-item aggregate model approach will be able to solve those problems.

The dynamic programming approach for a single-item model is used to search optimum solutions from those aggregate problems. The solutions will be used as benchmarks to evaluate the results from the heuristic programs. The data from *LDATA1 and *LDATA4 are used as testing data. The holding costs and the required quantity for discounts of both items are set to $2.00/unit/period and 200 units. Five different ordering costs are used to perform different tests. Each single test will carry 50 sets of data. The average results of these 50 sets of solutions are listed in Table 6-3.

Comparison Tests Among the Heuristic Programs

For other than the two selected situations we have just discussed, optimum solutions for most situations will require a costly search. As previously estimated, it will take more than 16,000 seconds to search for an optimum solution of a 12-period two-item single discount problem. We can hardly afford such costly tests. Instead we can carry some performance tests among the heuristic programs themselves. Again, the data from *LDATA1 and *LDATA4 were used as testing data. The situations with different ordering costs, and different discount rates were tested. The average cost of results generated by these three heuristic programs at different situations are listed in Table 6-6 and Table 6-7 for comparisons.

Discount Rate	Ordering Cost/ Holding Cost	PRG-1	PRG-2	PRG-3
$1	$300/$2	2648.96	2004.80	1987.32
	$206/$2	1919.76	1299.86	1265.76
	$120/$2	1041.84	585.18	532.96
	$ 92/$2	627.42	356.18	271.46
	$ 48/$2	-23.24	-57.58	-148.78
$2	$300/$2	1150.48	495.20	252.80
	$206/$2	638.44	-137.84	-377.64
	$120/$2	-280.84	-766.96	-978.76
	$ 92/$2	-540.08	-989.96	-1183.36
	$ 48/$2	-827.04	-1349.52	-1538.48
$3	$300/$2	-476.22	-1192.02	-1556.36
	$206/$2	-836.72	-1767.42	-2240.68
	$120/$2	-1262.90	-2390.26	-2841.90
	$ 92/$2	-1771.96	-2584.56	-3077.58
	$ 48/$2	-2338.38	-2925.90	-3430.46
$4	$300/$2	-2063.64	-3042.92	-3454.92
	$206/$2	-2437.36	-3665.52	-4106.40
	$120/$2	-2329.80	-4246.80	-4737.52
	$ 92/$2	-2737.04	-4444.88	-4952.56
	$ 48/$2	-3493.88	-4766.72	-5307.32

Table 6-6. Comparison Tests Among the Heuristic Programs Using *LDATA1 as Testing Data

Discount Rate	Ordering Cost/ Holding Cost	PRG-1	PRG-2	PRG-3
$1	$300/$2	2644.38	2039.36	2006.20
	$206/$2	2116.00	1335.18	1300.12
	$120/$2	1174.36	611.72	556.06
	$ 92/$2	800.52	375.30	289.66
	$ 48/$2	48.84	-32.96	-145.42
$2	$300/$2	1061.88	402.52	359.56
	$206/$2	583.16	-233.24	-377.84
	$120/$2	-14.12	-851.52	-1025.56
	$ 92/$2	-286.24	-1057.84	-1242.24
	$ 48/$2	-834.52	-1427.12	-1567.12
$3	$300/$2	-572.22	-1271.62	-1480.30
	$206/$2	-725.02	-1894.72	-2160.14
	$120/$2	-926.66	-2471.54	-2819.46
	$ 92/$2	-1512.00	-2683.44	-3056.14
	$ 48/$2	-1759.34	-3002.22	-3419.96
$4	$300/$2	-1921.76	-3023.16	-3363.24
	$206/$2	-2201.76	-3654.24	-4030.92
	$120/$2	-2264.68	-4235.76	-4681.28
	$ 92/$2	-2593.76	-4477.72	-4917.32
	$ 48/$2	-2838.12	-4847.60	-5276.44

Table 6-7. Comparison Tests Among the Heuristic Programs Using *LDATA4 as Testing Data

(a) Program-1 vs Program-3:

Discount/Unit	Ordering Cost/ Holding Cost $300/$2	$206/$2	$120/$2	$ 92/$2	$ 48/$2	Marginal Avg.
$1	649.91	734.94	563.59	433.41	159.90	508.35
$2	800.00	988.54	854.68	799.64	722.02	832.98
$3	994.11	1419.54	1735.90	1424.88	1376.35	1390.16
$4	1416.38	1749.10	2410.66	2269.54	2125.88	1994.31
Marginal Avg.	965.10	1223.03	1391.21	1231.86	1096.04	1181.45

(b) Program-2 vs Program-3:

Discount/Unit	Ordering Cost/ Holding Cost $300/$2	$206/$2	$120/$2	$ 92/$2	$ 48/$2	Marginal Avg.
$1	25.32	34.58	53.94	85.18	101.83	60.17
$2	142.68	192.20	192.92	188.90	164.48	176.24
$3	286.51	369.34	399.78	432.86	461.15	389.93
$4	376.04	408.78	466.62	473.64	484.72	441.96
Marginal Avg.	207.64	251.23	278.31	295.15	303.05	267.08

Table 6-8. Comparison Table of the Average Costs of Solutions Reached by Program-1 and Program-2 Over the Average Costs of Solutions Reached by Program-3

Holding Cost: both $2/unit/period
Qty. Req. for Discount: both 200 units
Discount Rate: $1

Ordering Cost	PRG-1	PRG-2	PRG-3
$300	0.050	0.062	0.054
$206	0.050	0.064	0.054
$120	0.052	0.066	0.054
$ 92	0.054	0.064	0.052
$ 48	0.054	0.064	0.056
Over All Average	0.052	0.064	0.054

Table 6-9. Average Required CPU Time for Heuristic Programs to Reach an Solution for a 12-Period Two-Item Single Discount Problem

Ordering Cost: $120/order

Holding Cost: both $2/unit/period

Qty. Req. for Discount: both 200 units

Discount Rate: $1/unit

Number of Data: 50 sets

No. of Periods	CPU Time First Run	Second Run	Avg. Time/Set
5	1.5 sec.	1.5 sec.	0.030 sec.
6	1.6 sec.	1.6 sec.	0.032 sec.
7	2.0 sec.	1.8 sec.	0.038 sec.
8	1.9 sec.	2.1 sec.	0.040 sec.
9	2.0 sec.	2.1 sec.	0.041 sec.
10	2.3 sec.	2.2 sec.	0.045 sec.
11	2.4 sec.	2.3 sec.	0.047 sec.
12	2.8 sec.	2.6 sec.	0.054 sec.

Table 6-10. Required CPU Time for Program-3 to Reach an Solution for a Multiperiod Two-Item Single Discount Problem

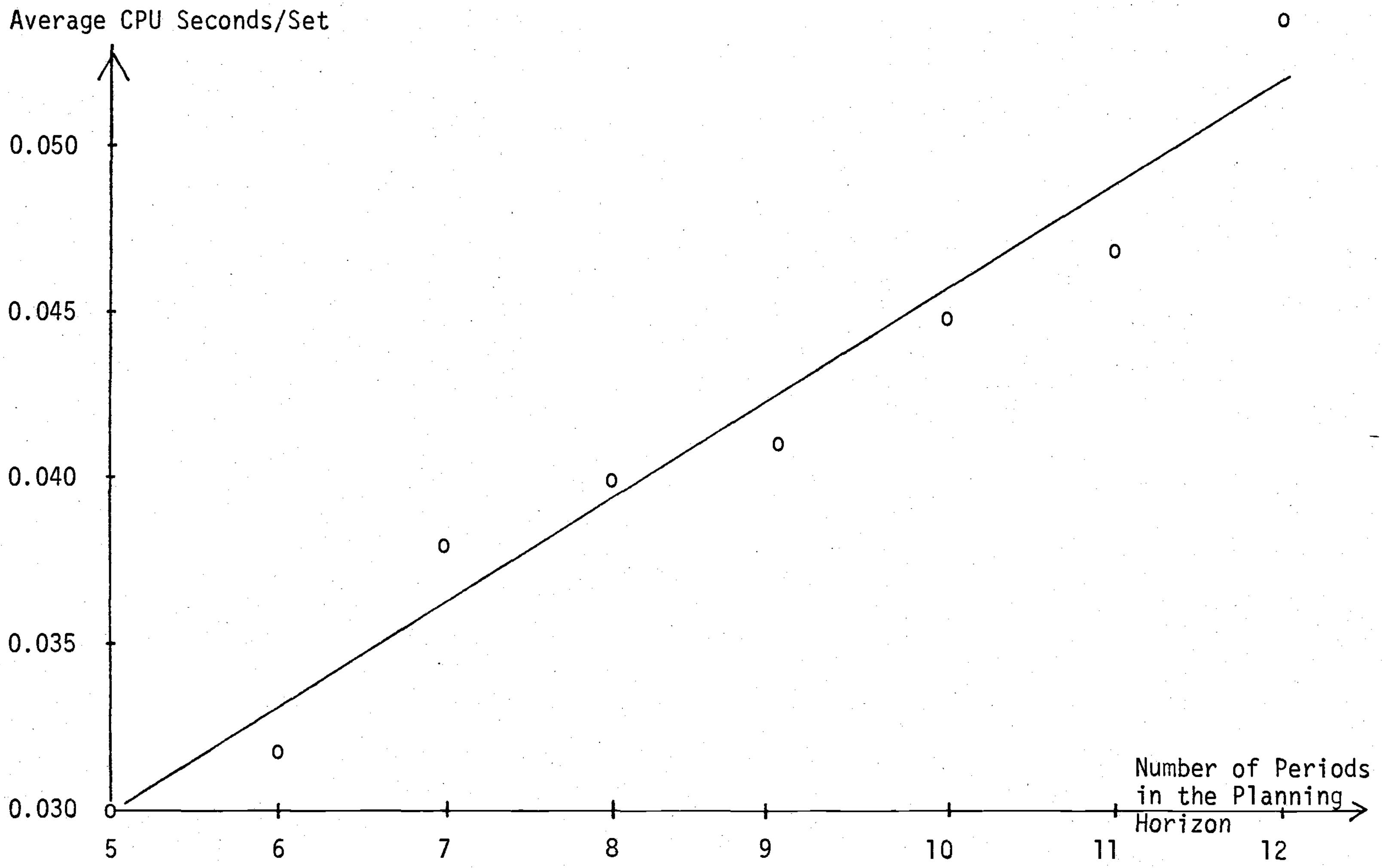


Figure 6-2 Required CPU Time for Program-3 to Reach a Solution for a Multiperiod Two-Item Single Discount Problem

In order to evaluate the test results more appropriately, the CPU time required by each program to reach test results was recorded in CDC 3300 CPU second. The average time from the 50 sets of data in *LDATA1 is listed in Table 6-9.

Evaluation of Test Results

Table 6-2 summarizes the results of performance tests when there is no orderind cost. The parameter chosen to vary in the tests is the discount rate. The overall result shows that Program-3, the one based on the Inoue-Chang Method, yielded results closest to the optimul solutions. The Program-2, based on the Silver-Meal Method, showed the second best overall results among the heuristic programs. The Program-1 was the worst. These results are similar to the ones obtained from the single-item problems. One interesting observation is that the average cost of Program-3 using the data group "LDATA1 is exactly the same as the results from using the data group "LDATA4. The reason for this is that Program-3 starts by comparing each period's holding cost to the ordering cost. Later, the Program-3 checks each item to see if it has reached the discount requirement. When there is no ordering cost, the holding cost is never less than the ordering cost. The tentative order will alwyas be placed at the next period. Then each item will be checked individually. Since "LDATA 1 and *LDATA4 are the same data arranged in different orders, Program-3 is expected to bring the same average costs from the two groups of data. Such characteristics do not exist in Program-1 and

Program-2 where the decisions are based on the minimum of unit cost and the periodic costs, and those costs include the discount savings which depend on the order quantity. Although apparently close, the results from each of these two programs using *LDATA1 and *LDATA4 are significantly different.

Table 6-3 is the summary results when there is no discount available. The tests search solutions of a multiple-item problem using the aggregate single-item model. The latter is only a minor modification of a single-item model. The results are similar to the results obtained by single-item problems (Table 3-6). Only the multiple-item results from Program-2 and Program-3 are closer than the single-item results from the Inoue-Chang and the Silver-Meal Methods. The reason is that during the development of Program-3, many considerations are eliminated in order to reduce the time-consuming search. The effect was to lower the searching time, while increasing the penalty costs of results. However, the results from Program-3 are still superior to the results from the other two programs in every category. Table 6-4 and 6-5 show the summary of the comparison of results from the heuristic programs using the results obtained by the dynamic programming approaches as the benchmark.

Except in some special cases, the search for an optimum solution in a multiple-item discount problem is time-consuming and costly. Table 6-6 and Table 6-7 summarize the comparison tests for those situations among various heuristic programs. Program-3 has the best overall performance in comparison with the other two programs; Program-2, the one basically developed from the Silver-Meal Method has the second best results. Those conclusions are similar to the ones obtained in the single-item cases. During the development of the Program-3, however, the computations were simplified by eliminating some procedures (like backtrack searches) to reduce the searching time. The Table 6-9 is the summary of the average CPU time each program needs to search for a solution. Program-1 and Program-3 take about the same amount of CPU time, while Program-2 takes the longest time among the three heuristic programs. We may thus conclude that Program-3, the one developed from the Inoue-Chang Method, has the best overall performance of all procedures investigated.

CHAPTER VII

CONCLUSION

Summary of the Study

The study of dynamic lot sizing problems under the discount situation is an area that has received relatively little attention from the previous studies and neglected by researchers in the production and inventory control fields. In modeling the problem at the single-item level, this thesis extended Wagner-Whitin's Planning Horizon Theorem to discount situations. The author further proved that this theorem represents the optimum algorithm search for a solution using the dynamic programming approach.

Along with other traditional heuristic approaches, a new approach, named Inoue-Chang Method, was proposed. The performance tests, using Kaimann's data and 100 additional sets of randomly generated data, showed that the results from the Inoue-Chang Method are generally superior to all other heuristic methods. The Inoue-Chang Method brought optimum results in 100% of the cases when performing tests using Kaimann's data, and an average of 97.4% of the cases when performing tests using the 100 additional sets of randomly generated data. The solutions were, on the average, 0.053% higher than the optimal solution using the second set of data. When performing tests using the same 100 sets of data, the Silver-Meal Method, and the Least Unit Cost Method brought optimum solutions in an average of 81.2% and 23.4% of

Single-Item Model	No Discount	Heuristic Program	Inoue-Chang Method
	Discount	Heuristic Program	Modified LUC, SM, and IC Methods
		Optimum Algorithm	Modified Planning Horizon Theorem

Multiple-Item Model	No Discount	Aggregate Single-Item Model		
	Discount	Heuristic Program	Program-1, Program-2, and Program-3	
		Optimum Algorithm	No Split Order	Extended Planning Horizon Theorem
			With Split Order	Mixed Integer Programming Model

Figure 7-1 Summary of the Products from This Thesis

the cases respectively, and cost .584% and 22.17% over the optimum solutions on the average.

The Least Unit Cost Method, Silver-Meal Method, and Inoue-Chang Method were separately modified in order to deal with the discount situations. Performance tests were carried using these methods in different discount situations. The results are summarized in Table 3-7 to Table 3-10 and Figure 3-8 to Figure 3-11. Again, it was found that, on the average, the Inoue-Chang Method brought the results closest to the optimum solutions, and was generally superior to both the Silver-Meal and Least Unit Cost Methods.

Based on the Least Unit Cost Method, Silver-Meal Method, and Inoue-Chang Method, three heuristic programs, Program-1, Program-2, and Program-3 were developed to search for solutions in the multiple-item discount situations. Both the multiple-item no discount situation and the multiple-item with discount situation were studied. The Planning Horizon Theorem was also extended to the multiple-item discount situations. A mixed integer programming model was developed for the situations when split orders were allowed. Two approaches, the Gomory's All Integer Algorithm, and the Branch-and-Bound Method attempted to search for an optimum solution from the model, and both failed. The difficulties involved were discussed and pointed out. Detailed information on the mixed integer programming model used in the two-item dynamic lot sizing problem is listed in Table 5-1. The required core memory sizes to search for an optimum solution from the model are listed in Table 5-1 and Figure 5-2.

Two-item problems with single discount levels were selected to illustrate the developed programs. The computer CPU time to search for an optimum solution involving different number of periods were measured and estimated. For a 12-period two-item single discount problem, it was estimated that it would require more than 16,000 CPU seconds to search for an optimum solution. To avoid such costly and time-consuming tests, two special cases were selected. These cases allowed us to use a single-item model to approximate a two-item model problem in searching for the optimum solutions. In other general situations, the performance tests were carried among the heuristic programs themselves. The heuristic Program-3, the one developed from the Inoue-Chang Method, again showed superior average results over the other two heuristic programs. Summarized results were listed in Table 6-6 to Table 6-7. In order to justify the comparisons, the required CPU time to solve a 12-period two-item single discount problem was measured for each heuristic program. The average CPU times were 0.052 second, 0.064 second, and 0.054 second for the three heuristic programs respectively.

Recommendations for Further Studies

The researches and studies on dynamic lot sizing problems started very late compared to the studies in other areas in the production and inventory control fields. There are a number of potential areas for further studies. Several of them are suggested.

Methodology to Search for an Optimum or a Near-Optimum Solution

Up to now, the dynamic programming approach is the only accepted method to search for an optimum solution for a dynamic lot sizing problem. But this method, as shown in this thesis, becomes time-consuming and costly in searching for an optimum solution when the number of periods in the planning horizon increases. The integer programming method can work on a very limited size model when the assumption allows split orders. Generally speaking, the integer programming method rarely works for any problem where the problem size is large enough to be practical. We need a simpler methodology to search for an optimum solution, or even a near-optimum solution.

Methodologies for Comparison

In evaluating heuristic approaches in this area, most authors make use of methodologies developed by Kaimann and Berry (Kaimann, 1969; Berry, 1972). These tests use five sets of standard data as the testing data (Table 3-1, Table 3-2), with the dynamic programming approach used to create benchmarks. This thesis proposed an additional 100 sets of randomly generated data for testing the performance. More standard data are needed in addition to the five sets of Kaimann's data. At the same time, more benchmark results should be made available. When the situation becomes complicated like the multiple-item discount situations, the results from the dynamic programming approach are generally difficult to obtain. Some methods are needed to generate benchmarks to fit the framework of analysis.

Application Areas

Almost all previous works in the area of dynamic lot sizing problems were centered around the single-item no discount problem. Very few researchers studied other application areas. This thesis extends the application areas to the multiple-item discount problems. The constraint applied to the multiple-item problem is the joint order. There are other kinds of constraints, such as limited capacities for multiple items, that represent some other potential application areas for further studies.

Stochastic Situation

One consistant assumption of this thesis is the deterministic demand. This, in many cases, is not true. To ignore the uncertainties of information may lead to inappropriate decisions. The uncertainties may come from different sources. The true demand quantities may differ from the forecasted values. The time of the demands may be earlier or later than the forecasted time. Under such situations, many questions may be raised. "What will be the best way to make the lot sizing decisions?", "What kind of penalty factors should be considered?", etc. When the multiple-item and discounts are involved, these questions represent a complex and challenging area to be investigated.

Conclusion

The dynamic lot size problem is frequently encountered by industrial engineers in production and inventory control systems. This thesis studied this problem under the multiple-item and discount situations, and developed some ordering procedures to deal with such situations. For a user to adopt an appropriate technique for his dynamic lot sizing problem, the most important thing is to understand his system's operation and to recognize the logic involved in the system. In order to make a choice of ordering procedures, ranging from simple lot-by-lot techniques to more sophisticated optimizing procedures, the decision will largely depend upon the inventory cost performance and the computational efficiency.

To deal with a complex problem, such as a multiple-item dynamic lot sizing problem involving the discount factors, this author is in favor of heuristic approaches. They provide simpler and lower cost methods to solve the problems, while the optimum methods usually cost more than what most users wish to spend. In most real-life problems, the situations are stochastic. The deterministic models, such as the ones we have studied in this thesis, are often used to obtain guidelines for decision-making under uncertainties. The precise optimality of solutions in such situations is not required. The difficulty of choosing an appropriate technique to solve a particular industrial problem is often alleviated if similar situations have already been analyzed by a fellow industrial engineer. Hopefully, this thesis will help the user to make his choice when he encounters a situation similar to the ones that this thesis has studied.

BIBLIOGRAPHY

Andres, F. and Emmons, H. 1975. A Multiproduct Inventory System with Interactive Set-Up Costs. Management Science. Vol. 21, No. 9: 1055-1063.

Berry, W. 1972. Lot Sizing Procedures for Requirements Planning Systems: A Framework for Analysis. Production and Inventory Management. Vol. 13, No. 2:19-34.

Callarman, T. E. and Whybark, D. C. 1977. Purchase Quantity Discounts in an MRP Environment. Discussion Paper #72. School of Business. Indiana University.

Chang, C. and Inoue, M.S. 1977. A New Dynamic Ordering Rule for Material Requirements Planning. AIIE 1977 Systems Engineering Conference Proceedings. pp. 173-177.

Chern, C. M. H. 1974. A Multi-product Joint Ordering Model with Dependent Setup Cost. Management Science. Vol. 20, No. 9: 1081-1091.

Denzler, D. R. 1970. A Heuristic Production Lot Sizing Scheduling Model. AIIE Transaction. Vol. 2, No. 3:59-63.

Eisenhut, P.S. 1975. A Dynamic Lot Sizing Algorithm with Capacity Constraints. AIIE Transaction. Vol. 7, No. 2:170-176.

Fabrycky, W. J. and Banks, J. 1967. Procurement and Inventory Systems. New York, Reinhold. 239 p.

Fortuin, L. 1977. A Survey of Literature on Reordering of Stock Items for Production Inventories. International Journal of Production Research. Vol. 15, No. 1:87-105.

Gorenstein, S. 1970. Some Remarks on EOQ vs Wagner-Whitin. Production and Inventory Management. Vol. 11, No. 2:40-46.

Gleason, J. M. 1971 A Computational Variation of the Wagner-Whitin Algorithm: An Alternative to the EOQ. Production and Inventory Management. Vol. 12, No. 1:15-22.

Johnson, L. A. and Montgomery, D. C. 1974. Operations Research in Production Planning, Scheduling, and Inventory Control. New York, John Wiley & Sons. 525 p.

Kaimann, R. A. 1969. E.O.Q. vs Dynamic Programming - Which One to Use for Inventory Ordering. Production and Inventory Management. Vol. 10, No. 4:66-74.

Nocturne, D. J. 1973. Economic Ordering Frequency for Several Items Jointly Replenished. Management Science. Vol. 19, No. 9: 1093-1096.

Orlicky, J. 1975. Material Requirements Planning. New York, McGraw-Hill. 292 p.

Plossl, G. W. 1973. Manufacturing Control: The Last Frontier for Profits. Reston, Reston Pub. Co. 174 p.

Riggs, J. L. and Inoue, M. S. 1975. Introduction to Operations Research and Management Science: A General Systems Approach. New York, McGraw-Hill. 497 p.

Ruch, W. A. 1976. Economic Lot Sizing in Material Requirements Planning. APICS 19th Annual Conference Proceedings. pp. 90-94.

Shu, F. T. 1971. Economic Ordering for Two Items Jointly Replenished. Management Science. Vol. 17, No. 5:B-406-410.

Silver, E. A. 1975. Modifying the Economic Order Quantity (EOQ) to Handle Coordinated Replenishment of Two or More Items. Production and Inventory Management. Vol. 16, No. 3:26-38.

Silver, E. A. and Meal, H. C. 1973. A Heuristic for Selecting Lot Sizing Quantities for the Case of a Deterministic Time-Varying Demand Rate and Discrete Opportunities for Replenishment. Production and Inventory Management. Vol. 14, No. 2:52-65.

Wagner, H. M. and Whitin, T. M. 1958. Dynamic Version of the Economic Lot Size Models. Management Science. Vol. 5, No. 1: 89-96.

Whybark, D. C. 1977. Determining the Attractiveness of Quantity Discounts. Discussion Paper #71. School of Business. Indiana University.

Wight, O. W. 1974. Production and Inventory Management in the Computer Age. Boston, Cahners. 284 p.

Zoller, K. 1977. Deterministic Mult-Item Inventory Systems with Limited Capacity. Management Science. Vol. 24, No. 4:451-455.

PLEASE NOTE:

Appendices contain pages with computer print-out. Print is broken and indistinct. Filmed as received.

UNIVERSITY MICROFILMS.

APPENDIX A

RANDOMLY GENERATED DATA

```
                          DATA                               VAR

 105 134 134  75 105 105  60 105  90  90  30  72            .324
 102 170  85  68 102 102  51  51 119 119  85  51            .382
  96 112  32 112  48  48  32  96 144 144 112 129            .453
  49 163  98 114 130 114  16 130  98  49  81  63            .457
  91  61  61  15  91 151 106 121 151 106 121  30            .472
  69  14 138 124 138  55  97  41  83  69 138 139            .473
 119  60  75  60 134  45 134 134  15  60 149 120            .481
 161  60 141  60 161  40  80 100  60  80  40 122            .482
 123 105 105 158  35 140 123  18  53  53 123  69            .488
  84 100 117  67 100  67  33 167 167  84 100  19            .489
  33  17 117 134 134 167  67 117  50  84 117  68            .499
  30 105  45   0 149 149 105 105  75 134  90 118            .510
 158 126  63 111  95  47  16 126 158  63 111  31            .518
  85  34  68 119  34 102  85 153 102 153  17 153            .519
 107  18 107  36  53 107 178  71 160  53 107 108            .521
 129 100   0  86  72 129 144 115  86 115   0 129            .522
  19 136  78 116 115  97  19  58  39 136 174 117            .541
  56 131  94  75 187  56 112  94   0 112 150  38            .558
 111 142 142  79 158  79   0  95  47  63 158  31            .563
  34 119  68  68  85 119 119  17 170 136  17 153            .565
  46 153  61  31 138 153  46 138 153  61 107  18            .577
 100 121  80 201 121 141  20  60 121  80  20  40            .579
 173  52  86   0  69 173 155  86  52 121  69  69            .579
   0 114 133  76 152 133  95 114 152   0  95  41            .580
  61 151 121 121   0 151  76  76 151  76   0 121            .583
 109  91 109 145  91  36 109 127 181 107   0   0            .596
  98 112   0 126 140 126   0 140  14 140 112  97            .596
  33 148  82  99  65  49 165 132  16  16 148 151            .608
  53 178  71  53  71  36   0 160 143  89 160  91            .611
  18  54 109  36 127  91  91 163 181 163  54  18            .626
 131 169  94 150   0  75   0 131 112  56  37 150            .636
  16 128 160 144 160  96  32  96 160  48  64   1            .641
 136  17 170 119 119 153  51  68  68  17 170  17            .647
 138 158   0 197  99 118  59 138  20  39  59  80            .648
  92  18  92  92 184  18  92 184 166  18  92  57            .656
  96   0 144 160 144   0  96 144 144  16 112  49            .659
 148  49 115  66 148  16  16 115 165  99   0 168            .664
 133  95  57 114 171 133 191   0  19  76   0 116            .689
  36 125 160   0 143 143   0  36 143  89  53 177            .699
  60 141  20  20 181  20 201 141 121 100  40  60            .701
 109  72  54 163 109   0  54 181   0 163 163  37            .709
  44   0 177  44 199 111  66  88  44 155 155  22            .718
 221  49  49 172   0 147 147  98  74  74  49  25            .720
 158  20   0 197  99 118  10 138  59 158  39 119            .728
 197   0 138 178   0 158  59 118  79   0  99  79            .750
 142 170  85  85 170   0 170 142  28   0 113   0            .756
  61 123 123 205   0  41 143 143 102   0   0 164            .762
   0 161 207  69  45   0 161  46 115 184  46  70            .776
 140  18 175 175   0  53 158 123  18 175  18  52            .778
 135 180   0  23   0  45 113  90  90 180  45 204            .779
 178   0 107  36   0 125   0 143 178  36 143 159            .786
 121  40  40 100 121 201 201 181  40  40   0  20            .789
 181  20  60  20   0  60 201 100  20 121 121 201            .799
 181   0 201  80 131  60   0   0  80  40 161 121            .821
  42   0 104 167  21  42 208   0  42 146 167 166            .821
   0 161  92 207   0 161 161   0  23 161  46  93            .833
 130 217 173 173   0  65 152 130  43   0   0  22            .859
 243  81 135 162 108   0   0 108 162   0 106   0            .862
 193 166 111  55   0   0 166 193   0  28  28 165            .879
 106 128 191   0  21   0 213   0 128  21 106 191            .884
  92 230 207 184  46   0 161  46  46  69  23   1            .891
 213   0   0   0 149  85   0  43 149 106 213 147            .899
  45 180 113 180 203   0 113  23 203  45   0   0            .901
   0   0 189  81   0  54 108 135 135 162 241   0            .903
  24   71  212  118  212   24  189    0  141  115    0    0      .908
  22  173  130  195    0   22  108  217    0    0  195   43      .942
   0   55  193  166  111  276    0   55   55  166   28    0      .978
  24  235    0  235    0  141   94   71    0  141  164    0      .982
 176  151  151   75   25    0   75  251    0    0  201    0      .984
  63    0   63    0  189    0   95  253   32  253   63   94      .998
 233    0    0  174    0   87  116    0    0  204   87  204     1.008
   0    0   44  111  221    0  155  177    0  199  198    0     1.013
 173  173   35   69  173    0  207   35    0    0    0  240     1.014
  58   29  116   58    0  204  116  233    0  262   29    0     1.027
  45  180  158  226   90  180    0  226    0    0    0    0     1.038
   0    0   60    0   90  209  209   30    0   90  269  148     1.039
 249    0   28   55  249    0    0   83  166   28   55  192     1.042
 231  206    0    0   77   77    0   51   26  206  231    0     1.062
 138  193  111    0  276  111    0    0   55  221    0    0     1.073
  61  123    0    0    0  123  153  184  153    0    0  308     1.076
 255   57  170   57    0   28  283  113    0  142    0    0     1.099
 239    0    0    0    0  119  149  209  239    0    0  150     1.112
   0  168   72    0    0  144  240   24    0    0  216  241     1.113
 257  128    0    0  128  128    0  257    0    0    0  207     1.146
  30   30  299   90  239  239   30    0   60   88    0    0     1.154
  69    0  138    0   35  138  311    0    0    0  242  172     1.169
   0  113  170  283  283    0    0    0    0   85    0  171     1.211
   0    0  107  285  214  285    0  143    0    0    0   71     1.241
   0    0    0    0  134    0  234  335    0   67  134  201     1.252
 189    0  158    0    0    0  284    0  253    0  221    0     1.280
 234    0  268  268   67    0    0  234    0   33    0    1     1.299
   0  260    0  260   65    0   65    0    0  325    0  130     1.329
 207    0    0   69   35  311    0    0    0    0  138  345     1.398
  71    0   36  321    0    0  250  107  320    0    0    0     1.406
   0  170  170  340   85    0    0  340    0    0    0    0     1.445
 201  151    0  452  100    0    0   50    0  151    0    0     1.470
   0    0  258  368    0   37  332    0   74   36    0    0     1.532
   0    0    0   41    0   82    0   41  205  368    0  368     1.541
   0    0   92    0  276    0    0  322    0  368   46    1     1.551
   0    0    0  263    0    0  158  263    0    0  421    0     1.601
```

APPENDIX B

AGGREGATE MODELS

```
      PROGRAM MULLOT
      DIMENSION ID(2,20),IHCOST(2),IQDCT(2),ISAV(2),VAR(2)
      WRITE(61,1)
    1 FORMAT(1H1,≠  NUMBER OF DATA, ND=≠)
      INDATA=TTYIN(4HND= )
      WRITE(61,2)
    2 FORMAT(≠  NUMBER OF PERIOD, NP=≠)
      N=TTYIN(4HNP= )
   11 WRITE(61,3)
    3 FORMAT(≠  ORDERING COST, OC=≠)
      IOCOST=TTYIN(4HOC= )
      DO 10 IJ=1,2
      WRITE(61,4) IJ
    4 FORMAT(≠  FOR ITEM-≠,I1,≠ HOLDING COST, HC=≠)
      IHCOST(IJ)=TTYIN(4HHC= )
      WRITE(61,5)
    5 FORMAT(≠  REQ. QUANTITY FOR DISCT., DQ=≠)
      IQDCT(IJ)=TTYIN(4HDQ= )
      WRITE(61,6)
    6 FORMAT(≠  DISCT. SAVING RATE, DS=≠)
      ISAV(IJ)=TTYIN(4HDS= )
   10 CONTINUE
    7 REWIND 2
      WRITE(1,100)
  100 FORMAT(1H1,//,22X,≠DEMANDS≠,24X,≠VAR≠,4X,≠ALG-1≠,5X,
     1≠ALG-2≠,4X,≠ALG-3≠)
      JTCS=LTCS=LUCTS=ITCS=0
      DO 20 IW=1,INDATA
      DO 30 IY=1,2
      READ(2,8) (ID(IY,I),I=1,N),VAR(IY)
    8 FORMAT(2X,12(I3,1X),5X,F6.3)
   30 CONTINUE
      CALL WW(ID,N,IOCOST,IHCOST,ITC,IODCT,ISAV,IJ)
      ITCS=ITCS+ITC
      CALL SM(ID,N,IOCOST,IHCOST,LTC,IQDCT,ISAV)
      CALL LUC(ID,N,IOCOST,IHCOST,LUCT,IQDCT,ISAV)
      CALL IC(ID,N,IOCOST,IHCOST,JTC,IODCT,ISAV)
      JTCS=JTCS+JTC
      LUCTS=LUCTS+LUCT
      LTCS=LTCS+LTC
      WRITE(1,200) (ID(1,I),I=1,N),VAR(1)
  200 FORMAT(2X,12(I3,1X),1X,F5.3)
      WRITE(1,250) (ID(2,I),I=1,N),VAR(2),LUCT,LTC,JTC,ITC
  250 FORMAT(2X,12(I3,1X),1X,F5.3,4(4X,I5))
   20 CONTINUE
      WRITE(1,300) LUCTS,LTCS,JTCS,ITCS
  300 FORMAT(/,49X,≠--SUM--≠,4(2X,I7))
      WRITE(61,500)
  500 FORMAT(≠  ANY CHANGE ON ORDERING COST^ OC=0 FOR≠
     1≠ ENDING≠)
      IOCOST=TTYIN(4HOC= )
      IF(IOCOST.EQ.0) GO TO 9
      GO TO 7
    9 WRITE(61,600)
  600 FORMAT(≠  ANY CHANGE ON OTHER PARAMETERS^ CP=0 FOR ≠
     1≠ENDING≠)
      ICP=TTYIN(4HCP= )
      IF(ICP.NE.0) GO TO 11
      STOP
      END
      SUBROUTINE WW(ID.N,IOCOST,IHCOST,ITC.IQOCT,ISAV,IJ)
      DIMENSION INV(20,20),IQTY(20,20),ITCOST(20,20)
      DIMENSION IHCOST(2),IQOCT(2),ISAV(2),ID(2,20)
      IMIN=0
      ISTART=1
 4    IF(ID(1,ISTART)+ID(2,ISTART).NE.0) GO TO 3
      ISTART=ISTART+1
      GO TO 4
 3    K=ISTART
      DO 1 J=ISTART,N
      INV(J,J)=IOCOST+IMIN
      IF(ID(1,J)+ID(2,J).EQ.0) INV(J,J)=IMIN
      IMIN=99999
      DO 1 I=ISTART,J
      JS1=J-1
      IG=ID(1,J)*IHCOST(1)+ID(2,J)*IHCOST(2)
      IF(J.GT.I) INV(I,J)=INV(I,JS1)+IG*(J-I)
      ITCOST(I,J)=INV(I,J)
      IF(ITCOST(I,J).GE.IMIN) GO TO 1
      IMIN=ITCOST(I,J)
      IF(IQTY(I,J).GE.IQOCT(IJ)) ISTART=I
 1    CONTINUE
      ITC=IMIN
      RETURN
      END
      SUBROUTINE IC(IDD,N,IOCOST,IHCOST,JTC,IQDCT,ISAV)
      DIMENSION IQ(2,20),IHCOST(2),IQDCT(2),ISAV(2)
      DIMENSION IDD(2,20),ITTQ(2),ITDMY(2),ID(2,20)
      DIMENSION INV(20)
      DO 1  I=1,N
      ID(1,I)=IDD(1,I)
      ID(2,I)=IDD(2,I)
      INV(I)=0
      IQ(1,I)=0
    1 IQ(2,I)=0
      NP1=N+1
      ID(1,NP1)=0
      ID(2,NP1)=0
      L=0
    2 L=L+1
      IF(L.GT.N) GO TO 200
      IF(ID(1,L)+ID(2,L).EQ.0) GO TO 2
      KK=1
    5 LK=L+KK
      IF(LK.GT.N) GO TO 7
      INV(LK)=ID(1,LK)*KK*IHCOST(1)+ID(2,LK)*KK*IHCOST(2)
      IF(INV(LK).GE.IOCOST) GO TO 10
      KK=KK+1
      GO TO 5
   10 INV(LK)=0
      LKK=LK
    7 IF(KK.LE.2) GO TO 13
      LKK=LK-1
      ITTQ(1)=ID(1,LKK)
      ITTQ(2)=ID(2,LKK)
      IBACK=1
    6 IBACK=IBACK+1
      NKK=KK-IBACK
      IF(NKK.EQ.0) GO TO 13
      LKK=L+NKK
      ITTQ(1)=ITTQ(1)+ID(1,LKK)
      ITTQ(2)=ITTQ(2)+ID(2,LKK)
      IMGINV=(ITTQ(1)*IHCOST(1)+ITTQ(2)*IHCOST(2))*NKK
      IF(IMGINV.LT.IOCOST) GO TO 6
      INV(LKK)=0
      LK=LK
   13 IEND=LK-1
      DO 30 IJ=1,2
      IH=IT=0
      DO 20 I=L,IEND
   20 IT=IT+ID(IJ,I)
      ITDMY(IJ)=IT
      IF(IT.GE.IQDCT) GO TO 30
      IDMY=IEND+1
   27 IF(IDMY.GT.N) GO TO 30
      IH=IH+IHCOST(IJ)*ID(IJ,IDMY)*(LK-L)
      IT=IT+ID(IJ,IDMY)
      IF(IT.GE.IQDCT(IJ)) GO TO 25
      IDMY=IDMY+1
      GO TO 27
   25 ISAVING=IT*ISAV(IJ)
      IF(IT-ITDMY(IJ).GE.IQDCT(IJ)) ISAVING=ITDMY(IJ)
     1*ISAV(IJ)
      IF(ISAVING.LT.IH) GO TO 30
      ITDMY(IJ)=IT
      DO 45 I=LK,IDMY
   45 ID(IJ,I)=0
   30 CONTINUE
      IQ(1,L)=ITDMY(1)
      IQ(2,L)=ITDMY(2)
      L=LK-1
      GO TO 2
  200 CALL BACKTR(IQ,IOCOST,IHCOST,N)
      CALL COST(IQ,IOCOST,IHCOST,N,IDD,IQDCT,ISAV,JC,ITS)
      JTC=JC-ITS
      RETURN
      END
      SUBROUTINE SM(IDD,N,IOCOST,IHCOST,LTC,IQDCT,ISAV)
      DIMENSION ID(2,20),IDD(2,20),JQ(2,20),IHCOST(2)
      DIMENSION IDMY1(2),IDMY2(2),IG1(2),IG2(2),IOQ(2)
      DIMENSION IQDCT(2),ISAV(2)
      DO 1 I=1,N
      ID(1,I)=IDD(1,I)
    1 ID(2,I)=IDD(2,I)
      NP1=N+1
      ITC=0
      ID(1,NP1)=0
      ID(2,NP1)=0
      DO 2 IK=1,NP1
      JQ(1,IK)=0
    2 JQ(2,IK)=0
      IT=1
   30 IF(IT.GT.N) GO TO 1000
      IF(ID(1,IT)+ID(2,IT).GT.0) GO TO 5
      IT=IT+1
      GO TO 30
    5 IOQ(1)=0
      IOQ(2)=0
      IOT=IT
      J=1
    4 JP1=J+1
      IOQ(1)=ID(1,IT)+IOQ(1)
      IOQ(2)=ID(2,IT)+IOQ(2)
      IF(IT.EQ.NP1) GO TO 10
      IT=IT+1
      IA=J*J*(ID(1,IT)*IHCOST(1)+ID(2,IT)*IHCOST(2))
      IB=IOCOST
      DO 200 IJ=1,2
      IDMY1(IJ)=0
  200 IDMY2(IJ)=0
      DO 300 I=1,J
      IS1=I-1
      JK=IOT-1+I
      DO 400 IJ=1,2
      IDMY1(IJ)=IDMY1(IJ)+ID(IJ,JK)
  400 IB=IB+IS1*ID(IJ,JK)*IHCOST(IJ)
  300 CONTINUE
      JP1=J+1
      DO 500 IJ=1,2
      IDMY2(IJ)=IDMY1(IJ)+ID(IJ,JP1)
      IG1(IJ)=0
      IG2(IJ)=0
      IF(IDMY1(IJ).GE.IQDCT(IJ)) IG1(IJ)=ISAV(IJ)
      IF(IDMY2(IJ).GE.IQDCT(IJ)) IG2(IJ)=ISAV(IJ)
      IA=IA-J*IDMY2(IJ)*IG2(IJ)
  500 IB=IB-JP1*IDMY1(IJ)*IG1(IJ)
      IF(IA.GT.IB) GO TO 10
      J=J+1
      GO TO 4
   10 IJ=1
      IF(IT.GT.N) GO TO 14
      IF(IOQ(1).GE.IQDCT(1)) GO TO 14
      CALL SMCOM(J,IOT,IOCOST,IHCOST,ID,IT,IQDCT,ISAV,IOQ,
     1NP1,IJ)
   14 JQ(1,IOT)=IOQ(1)
      IJ=2
      IF(IT.GT.N) GO TO 15
      IF(IOQ(2).GE.IQDCT(2)) GO TO 15
      CALL SMCOM(J,IOT,IOCOST,IHCOST,ID,IT,IQDCT,ISAV,IOQ,
     1NP1,IJ)
   15 JQ(2,IOT)=IOQ(2)
      GO TO 30
 1000 CALL COST(JQ,IOCOST,IHCOST,N,IDD,IQDCT,ISAV,JC,ITS)
      LTC=JC-ITS
      RETURN
      END
      SUBROUTINE LUC(IDD,N,IOCOST,IHCOST,LUCT,IQDCT,ISAV)
      DIMENSION JO(2,20),IHCOST(2),IQDCT(2),ISAV(2),M(2)
      DIMENSION MTQ(2),IG1(2),IG2(2),IDD(2,20),ID(2,20)
      DO 5 J=1,2
      DO 1 I=1,N
      JO(J,I)=0
      ID(J,I)=IDD(J,I)
    1 CONTINUE
    5 CONTINUE
      I=1
    2 IF(ID(1,I)+ID(2,I).NE.0) GO TO 30
      I=I+1
      IF(I.GT.N) GO TO 98
      GO TO 2
   30 K=I
      M(1)=ID(1,I)
      M(2)=ID(2,I)
      IG1(1)=0
      IG1(2)=0
      IG2(1)=0
      IG2(2)=0
      J=1
      IF(M(1).GE.IQDCT(1)) IG1(1)=ISAV(1)
      IF(M(2).GE.IQDCT(2)) IG1(2)=ISAV(2)
      IFT=IOCOST-IG1(1)*M(1)-IG1(2)*M(2)
   10 MTQ(1)=M(1)
      MTQ(2)=M(2)
      I=I+1
      IF(I.GT.N) GO TO 98
      M(1)=M(1)+ID(1,I)
      M(2)=M(2)+ID(2,I)
      IF(M(1).GE.IQDCT(1)) IG2(1)=ISAV(1)
      IF(M(2).GE.IQDCT(2)) IG2(2)=ISAV(2)
      IFTP1=IFT+ID(1,I)*IHCOST(1)*J+ID(2,I)*IHCOST(2)*J
      IFTP1=IFTP1-IG2(1)*M(1)-IG2(2)*M(2)+IG1(1)*MTQ(1)
     1+IG1(2)*MTQ(2)
      MTQN=MTQ(1)+MTQ(2)
      MN=M(1)+M(2)
      IF(IFTP1*MTQN.GT.IFT*MN) GO TO 50
      IG1(1)=IG2(1)
      IG1(2)=IG2(2)
      J=J+1
      IFT=IFTP1
      GO TO 10
   50 IK=1
      I1=99999
      IF(M(IK).LT.IQDCT(IK)) GO TO 60
      GO TO 100
   60 IST=I
      IH=0
      L=J
      MG=M(IK)
   64 IST=IST+1
      L=L+1
      IF(IST.LE.N) GO TO 62
      IF(IK.EQ.1) GO TO 100
      GO TO 98
   62 MG=MG+ID(IK,IST)
      IH=IH+ID(IK,IST)*IHCOST(IK)*L
      IF(MG.LT.IQDCT(IK)) GO TO 64
      MTOTAL=M(1)+M(2)
      IFTP2=IFTP1+IH-ISAV(IK)*M(IK)
      IF(IFTP2*MTQN.LE.IFT*MTOTAL) GO TO 120
      IF(IK.EQ.1) GO TO 100
      GO TO 98
  120 DO 130 JK=I,IST
  130 ID(IK,JK)=0
      MTQ(IK)=MG
      M(IK)=MG
      IF(IK.EQ.2) GO TO 98
```

```
100 IK=2
    IF(M(IK).LT.IODCT(IK)) GO TO 60
 98 JQ(1,K)=MTQ(1)
    JQ(2,K)=MTQ(2)
    GO TO 2
 99 CALL COST(JQ,IOCOST,IHCOST,N,ID,IODCT,ISAV,JC,ITS)
    LUCT=JC-ITS
    RETURN
    END
```

```
      SUBROUTINE BACKTR(IQ,IOCOST,IHCOST,N)
      DIMENSION IQ(2,20),IHCOST(2)
      J=N
  200 IF(IQ(1,J)+IQ(2,J).EQ.0) GO TO 100
      IF(IQ(1,J).GT.0.AND.IQ(2,J).GT.0) GO TO 100
      K=0
      JT=J
      JQ=IQ(1,J)*IHCOST(1)+IQ(2,J)*IHCOST(2)
  150 K=K+1
      IF(JQ*K.GT.IOCOST) GO TO 100
      J=J-1
      IF(J.EQ.0) GO TO 1000
      IF(IQ(1,J)+IQ(2,J).EQ.0) GO TO 150
      IQ(1,J)=IQ(1,J)+IQ(1,JT)
      IQ(2,J)=IQ(2,J)+IQ(2,JT)
      IQ(1,JT)=0
      IQ(2,JT)=0
  100 J=J-1
      IF(J.EQ.0) GO TO 1000
      GO TO 200
 1000 RETURN
      END


      SUBROUTINE SMCOM(J,IOT,IOCOST,IHCOST,ID,IT,IQDCT,ISAV,
     1IOQ,NP1,IJ)
      DIMENSION ID(2,20),IOQ(2),IQDCT(2),IHCOST(2),ISAV(2)
      JTEMP=J
      IH=0
      IQ=IOQ(IJ)
      IT1=IT
   20 IF(IT1.EQ.NP1) GO TO 10
      JTEMP=JTEMP+1
      IQ=IQ+ID(IJ,IT1)
      IH=IH+ID(IJ,IT1)*IHCOST(IJ)*J
      IT1=IT1+1
      IF(IQ.LT.IQDCT(IJ)) GO TO 20
      JQ=IQ
      IF(IQ-IOQ(IJ).GE.IQDCT(IJ)) JQ=IOQ(IJ)
      IDCT=JQ*ISAV(IJ)
      IF(IDCT.LT.IH) GO TO 10
      IOQ(IJ)=IQ
      IZ=IT1-1
      DO 30 I=IT,IZ
   30 ID(IJ,I)=0
   10 RETURN
      END


      SUBROUTINE COST(JQ,IO,IH,N,ID,IQDCT,ISAV,JC,IIS)
      DIMENSION JQ(2,20),IH(2),ID(2,20),IQDCT(2),ISAV(2)
      JO=JH=ITS=L1=L2=0
      DO 100 I=1,N
      IF(JQ(1,I)+JQ(2,I).GT.0) JO=JO+IO
      IF(JQ(1,I).GE.IQDCT(1)) ITS=IIS+JQ(1,I)*ISAV(1)
      IF(JQ(2,I).GE.IQDCT(2)) ITS=IIS+JQ(2,I)*ISAV(2)
      L1=L1+JQ(1,I)-ID(1,I)
      L2=L2+JQ(2,I)-ID(2,I)
      JH=JH+L1*IH(1)+L2*IH(2)
  100 CONTINUE
      JC=JH+JO
      RETURN
      END
```

APPENDIX C

THREE HEURISTIC PROGRAMS

```
      PROGRAM MULLOT
      DIMENSION ID(2,20),IHCOST(2),IQDCT(2),ISAV(2)
      DIMENSION VAR(2)
      WRITE(61,1)
    1 FORMAT(1H1,≠  NUMBER OF DATA, ND=≠)
      INDATA=TTYIN(4HND= )
      WRITE(61,2)
    2 FORMAT(≠  NUMBER OF PERIOD, NP=≠)
      N=TTYIN(4HNP= )
   11 WRITE(61,3)
    3 FORMAT(≠  ORDERING COST, OC=≠)
      IOCOST=TTYIN(4HOC= )
      DO 10 IJ=1,2
      WRITE(61,4) IJ
    4 FORMAT(≠  FOR ITEM-≠,I1,≠ HOLDING COST, HC=≠)
      IHCOST(IJ)=TTYIN(4HHC= )
      WRITE(61,5)
    5 FORMAT(≠  REQ. QUANTITY FOR DISCT., DQ=≠)
      IQDCT(IJ)=TTYIN(4HDQ= )
      WRITE(61,6)
    6 FORMAT(≠  DISCT. SAVING RATE, DS=≠)
      ISAV(IJ)=TTYIN(4HDS= )
   10 CONTINUE
    7 REWIND 2
      WRITE(1,100)
  100 FORMAT(1H1,//,22X,≠DEMANDS≠,24X,≠VAR≠,4X,
     1≠ALG-1≠,5X,≠ALG-2≠,4X,≠ALG-3≠)
      JTCS=LTCS=LUCTS=0
      DO 20 IW=1,INDATA
      DO 30 IY=1,2
      READ(2,8) (ID(IY,I),I=1,N),VAR(IY)
    8 FORMAT(2X,12(I3,1X),5X,F6.3)
   30 CONTINUE
      CALL SM(ID,N,IOCOST,IHCOST,LTC,IQDCT,ISAV)
      CALL LUC(ID,N,IOCOST,IHCOST,LUCT,IQDCT,ISAV)
      CALL IC(ID,N,IOCOST,IHCOST,JTC,IQDCT,ISAV)
      JTCS=JTCS+JTC
      LUCTS=LUCTS+LUCT
      LTCS=LTCS+LTC
      WRITE(1,200) (ID(1,I),I=1,N),VAR(1)
  200 FORMAT(2X,12(I3,1X),1X,F5.3)
      WRITE(1,250) (ID(2,I),I=1,N),VAR(2),LUCT,LTC,JTC
  250 FORMAT(2X,12(I3,1X),1X,F5.3,3(4X,I5))
   20 CONTINUE
      WRITE(1,300) LUCTS,LTCS,JTCS
  300 FORMAT(/,49X,≠--SUM--≠,3(2X,I7))
      WRITE(61,500)
  500 FORMAT(≠  ANY CHANGE ON ORDERING COST^ OC=0 ≠
     1≠FOR ENDING≠)
      IOCOST=TTYIN(4HOC= )
      IF(IOCOST.EQ.0) GO TO 9
      GO TO 7
    9 WRITE(61,600)
  600 FORMAT(≠  ANY CHANGE ON OTHER PARAMETERS^ CP=0 ≠
     1≠FOR ENDING≠)
      ICP=TTYIN(4HCP= )
      IF(ICP.NE.0) GO TO 11
      STOP
      END
      SUBROUTINE IO(IDD,N,IOCOST,IHCOST,JTC,IQDCT,ISAV)
      DIMENSION IQ(2,20),IHCOST(2),IQDCT(2),ISAV(2)
      DIMENSION IDD(2,20),ITTQ(2),ITDMY(2),ID(2,20),INV(20)
      DO 1 I=1,N
      ID(1,I)=IDD(1,I)
      ID(2,I)=IDD(2,I)
    1 INV(I)=IQ(1,I)=IQ(2,I)=0
      NP1=N+1
      ID(1,NP1)=0
      ID(2,NP1)=0
      L=0
    2 L=L+1
      IF(L.GT.N) GO TO 200
      IF(ID(1,L)+ID(2,L).EQ.0) GO TO 2
      KK=1
    5 LK=L+KK
      IF(LK.GT.N) GO TO 7
      INV(LK)=ID(1,LK)*KK*IHCOST(1)+ID(2,LK)*KK*IHCOST(2)
      IF(INV(LK).GE.IOCOST) GO TO 10
      KK=KK+1
      GO TO 5
   10 INV(LK)=0
      LKK=LK
    7 IF(KK.LE.2) GO TO 13
      LKK=LK-1
      ITTQ(1)=ID(1,LKK)
      ITTQ(2)=ID(2,LKK)
      IBACK=1
    6 IBACK=IBACK+1
      NKK=KK-IBACK
      IF(NKK.EQ.0) GO TO 13
      LKK=L+NKK
      ITTQ(1)=ITTQ(1)+ID(1,LKK)
      ITTQ(2)=ITTQ(2)+ID(2,LKK)
      IMGINV=(ITTQ(1)*IHCOST(1)+ITTQ(2)*IHCOST(2))*NKK
      IF(IMGINV.LT.IOCOST) GO TO 6
      INV(LKK)=0
      LK=LK
   13 IEND=LK-1
      DO 30 IJ=1,2
      IH=IT=0
      DO 20 I=L,IEND
   20 IT=IT+ID(IJ,I)
      ITDMY(IJ)=IT
      IF(IT.GE.IQDCT) GO TO 30
      IDMY=IEND+1
   27 IF(IDMY.GT.N) GO TO 30
      IH=IH+IHCOST(IJ)*ID(IJ,IDMY)*(LK-L)
      IT=IT+ID(IJ,IDMY)
      IF(IT.GE.IQDCT(IJ)) GO TO 25
      IDMY=IDMY+1
      GO TO 27
   25 ISAVING=IT*ISAV(IJ)
      IF(IT-ITDMY(IJ).GE.IQDCT(IJ))
     1ISAVING=ITDMY(IJ)*ISAV(IJ)
      IF(ISAVING.LT.IH) GO TO 30
      ITDMY(IJ)=IT
      DO 45 I=LK,IDMY
   45 ID(IJ,I)=0
   30 CONTINUE
      IQ(1,L)=ITDMY(1)
      IQ(2,L)=ITDMY(2)
      L=LK-1
      GO TO 2
  200 CALL BACKTR(IQ,IOCOST,IHCOST,N)
      CALL COST(IQ,IOCOST,IHCOST,N,IDD,IQDCT,ISAV,JC,ITS)
      JTC=JC-ITS
      RETURN
      END
      SUBROUTINE SM(IDD,N,IOCOST,IHCOST,LTC,IQDCT,ISAV)
      DIMENSION ID(2,20),IDD(2,20),JQ(2,20),IQDCT(2)
      DIMENSION IDMY1(2),IDMY2(2),IG1(2),IG2(2),IOQ(2)
      DIMENSION IHCOST(2),ISAV(2)
      DO 1 I=1,N
      ID(1,I)=IDD(1,I)
      ID(2,I)=IDD(2,I)
    1 CONTINUE
      NP1=N+1
      ITC=0
      ID(1,NP1)=0
      ID(2,NP1)=0
      DO 2 IK=1,NP1
      JQ(1,IK)=0
    2 JQ(2,IK)=0
      IT=1
   30 IF(IT.GT.N) GO TO 1000
      IF(ID(1,IT)+ID(2,IT).GT.0) GO TO 5
      IT=IT+1
      GO TO 30
    5 IOQ(1)=0
      IOQ(2)=0
      IOT=IT
      J=1
    4 JP1=J+1
      IOQ(1)=ID(1,IT)+IOQ(1)
      IOQ(2)=ID(2,IT)+IOQ(2)
      IF(IT.EQ.NP1) GO TO 10
      IT=IT+1
      IA=J*J*(ID(1,IT)*IHCOST(1)+ID(2,IT)*IHCOST(2))
      IB=IOCOST
      DO 200 IJ=1,2
      IDMY1(IJ)=0
  200 IDMY2(IJ)=0
      DO 300 I=1,J
      IS1=I-1
      JK=IOT-1+I
      DO 400 IJ=1,2
      IDMY1(IJ)=IDMY1(IJ)+ID(IJ,JK)
  400 IB=IB+IS1*ID(IJ,JK)*IHCOST(IJ)
  300 CONTINUE
      JP1=J+1
      DO 500 IJ=1,2
      IDMY2(IJ)=IDMY1(IJ)+ID(IJ,JP1)
      IG1(IJ)=0
      IG2(IJ)=0
      IF(IDMY1(IJ).GE.IQDCT(IJ)) IG1(IJ)=ISAV(IJ)
      IF(IDMY2(IJ).GE.IQDCT(IJ)) IG2(IJ)=ISAV(IJ)
      IA=IA-J*IDMY2(IJ)*IG2(IJ)
  500 IB=IB-JP1*IDMY1(IJ)*IG1(IJ)
      IF(IA.GT.IB) GO TO 10
      J=J+1
      GO TO 4
   10 IJ=1
      IF(IT.GT.N) GO TO 14
      IF(IOQ(1).GE.IQDCT(1)) GO TO 14
      CALL SMCOM(J,IOT,IOCOST,IHCOST,ID,IT,IQDCT,ISAV,
     1IOQ,NP1,IJ)
   14 JQ(1,IOT)=IOQ(1)
      IJ=2
      IF(IT.GT.N) GO TO 15
      IF(IOQ(2).GE.IQDCT(2)) GO TO 15
      CALL SMCOM(J,IOT,IOCOST,IHCOST,ID,IT,IQDCT,ISAV,
     1IOQ,NP1,IJ)
   15 JQ(2,IOT)=IOQ(2)
      GO TO 30
 1000 CALL COST(JQ,IOCOST,IHCOST,N,IDD,IQDCT,ISAV,JC,ITS)
      LTC=JC-ITS
      RETURN
      END
      SUBROUTINE LUC(IDD,N,IOCOST,IHCOST,LUCT,IQDCT,ISAV)
      DIMENSION JQ(2,20),IHCOST(2),IQDCT(2),ISAV(2),M(2)
      DIMENSION MTQ(2),IG1(2),IG2(2),IDD(2,20),ID(2,20)
      DO 1  J=1,2
      DO 1  I=1,N
      JQ(J,I)=0
    1 ID(J,I)=IDD(J,I)
      I=1
    2 IF(ID(1,I)+ID(2,I).NE.0) GO TO 30
      I=I+1
      IF(I.GT.N) GO TO 99
      GO TO 2
   30 K=I
      M(1)=ID(1,I)
      M(2)=ID(2,I)
      IG1(1)=IG1(2)=IG2(1)=IG2(2)=0
      J=1
      IF(M(1).GE.IQDCT(1)) IG1(1)=ISAV(1)
      IF(M(2).GE.IQDCT(2)) IG1(2)=ISAV(2)
      IFT=IOCOST-IG1(1)*M(1)-IG1(2)*M(2)
   10 MTQ(1)=M(1)
      MTQ(2)=M(2)
      I=I+1
      IF(I.GT.N) GO TO 98
      M(1)=M(1)+ID(1,I)
      M(2)=M(2)+ID(2,I)
      IF(M(1).GE.IQDCT(1)) IG2(1)=ISAV(1)
      IF(M(2).GE.IQDCT(2)) IG2(2)=ISAV(2)
      IFTP1=IFT+ID(1,I)*IHCOST(1)*J+ID(2,I)*IHCOST(2)*J
      IFTP1=IFTP1-IG2(1)*M(1)-IG2(2)*M(2)+IG1(1)*MTQ(1)
     1+IG1(1)*MTQ(1)
      MTQN=MTQ(1)+MTQ(2)
      MN=M(1)+M(2)
      IF(IFTP1*MTQN.GT.IFT*MN) GO TO 50
      IG1(1)=IG2(1)
      IG1(2)=IG2(2)
      J=J+1
      IFT=IFTP1
      GO TO 10
   50 IK=1
      I1=99999
      IF(M(IK).LT.IQDCT(IK)) GO TO 60
      GO TO 100
   60 IST=I
      IH=0
      L=J
      MG=M(IK)
   64 IST=IST+1
      L=L+1
      IF(IST.LE.N) GO TO 62
      IF(IK.EQ.1) GO TO 100
      GO TO 98
   62 MG=MG+ID(IK,IST)
      IH=IH+ID(IK,IST)*IHCOST(IK)*L
      IF(MG.LT.IQDCT(IK)) GO TO 64
      MTOTAL=M(1)+M(2)
      IFTP2=IFTP1+IH-ISAV(IK)*M(IK)
      IF(IFTP2*MTQN.LE.IFT*MTOTAL) GO TO 120
      IF(IK.EQ.1) GO TO 100
      GO TO 98
  120 DO 130 JK=I,IST
  130 ID(IK,JK)=0
      MTQ(IK)=MG
      M(IK)=MG
      IF(IK.EQ.2) GO TO 98
  100 IK=2
      IF(M(IK).LT.IQDCT(IK)) GO TO 60
   98 JQ(1,K)=MTQ(1)
      JQ(2,K)=MTQ(2)
      GO TO 2
   99 CALL COST(JQ,IOCOST,IHCOST,N,ID,IQDCT,ISAV,JC,ITS)
      LUCT=JC-ITS
      RETURN
      END
      SUBROUTINE SMCOM(J,IOT,IOCOST,IHCOST,ID,IT,IQDCT,
     1ISAV,IOQ,NP1,IJ)
      DIMENSION ID(2,20),IOQ(2),IQDCT(2),IHCOST(2),ISAV(2)
      JTEMP=J
      IH=0
      IQ=IOQ(IJ)
      IT1=IT
   20 IF(IT1.EQ.NP1) GO TO 10
      JTEMP=JTEMP+1
      IQ=IQ+ID(IJ,IT1)
      IH=IH+ID(IJ,IT1)*IHCOST(IJ)*J
      IT1=IT1+1
      IF(IQ.LT.IQDCT(IJ)) GO TO 20
      JQ=IQ
      IF(IQ-IOQ(IJ).GE.IQDCT(IJ)) JQ=IOQ(IJ)
      IDCT=JQ*ISAV(IJ)
      IF(IDCT.LT.IH) GO TO 10
      IOQ(IJ)=IQ
      IZ=IT1-1
      DO 30 I=IT,IZ
   30 ID(IJ,I)=0
   10 RETURN
      END


      SUBROUTINE BACKTR(IQ,IOCOST,IHCOST,N)
      DIMENSION IQ(2,20),IHCOST(2)
      J=N
  200 IF(IQ(1,J)+IQ(2,J).EQ.0) GO TO 100
      IF(IQ(1,J).GT.0.AND.IQ(2,J).GT.0) GO TO 100
      K=0
      JT=J
      JQ=IQ(1,J)*IHCOST(1)+IQ(2,J)*IHCOST(2)
  150 K=K+1
      IF(JQ*K.GT.IOCOST) GO TO 100
      J=J-1
      IF(J.EQ.0) GO TO 1000
      IF(IQ(1,J)+IQ(2,J).EQ.0) GO TO 150
      IQ(1,J)=IQ(1,J)+IQ(1,JT)
      IQ(2,J)=IQ(2,J)+IQ(2,JT)
      IQ(1,JT)=0
      IQ(2,JT)=0
  100 J=J-1
      IF(J.EQ.0) GO TO 1000
      GO TO 200
 1000 RETURN
      END


      SUBROUTINE COST(JQ,IO,IH,N,ID,IQDCT,ISAV,JC,ITS)
      DIMENSION JQ(2,20),IH(2),ID(2,20),IQDCT(2),ISAV(2)
      JO=JH=ITS=L1=L2=0
      DO 100 I=1,N
      IF(JQ(1,I)+JQ(2,I).GT.0) JO=JO+IO
      IF(JQ(1,I).GE.IQDCT(1)) ITS=ITS+JQ(1,I)*ISAV(1)
      IF(JQ(2,I).GE.IQDCT(2)) ITS=ITS+JQ(2,I)*ISAV(2)
      L1=L1+JQ(1,I)-ID(1,I)
      L2=L2+JQ(2,I)-ID(2,I)
      JH=JH+L1*IH(1)+L2*IH(2)
  100 CONTINUE
      JC=JH+JO
      RETURN
      END
```