A Heuristic for Multi-Level Lot-Sizing Problems with Bottlenecks under a Rolling Schedule Environment.

by

Bilal Toklu
B.Sc., M.Sc.

A Doctoral Thesis
Submitted in partial fulfilment of the requirements for
the award of Doctor of Philosophy
February 1993

Supervisor: Senior Lecturer John M. Wilson B.Sc., M.Sc. DPhil

ACKNOWLEDGMENT

I would like to express my sincere appreciation and thanks to Senior Lecturer John M. Wilson who has acted as my supervisor and who made himself available constantly for consultation. His help, patience, comments, and suggestions on various topics have been greatly appreciated in the preparation of this thesis.

I would also like to thank my research panel under the directorship of Professor Geoff Gregory and Professor Malcolm King and panelist Senior Lecturer David Johnson. They made significant suggestions and criticism during research progress.

I would like to thank Professor P. J. Billington for making data available to us and providing encouragement and advice, and also the referees of the International Journal of Production Research for their valuable comments and criticism.

I would also like to thank Dr Paul D. Mizen, Mrs Janet Stevenson, and Mr. David W. Lamb for checking my work and providing encouragement.

My thanks also go to Dr Abdul Rahman Khan, Dr Mohammed A. Rahin for their help and advice about computer programming.

I would also like to acknowledge to my employers "Gazi University of Turkey" for their financial support during the period of this research. This has made it possible for me to pursue my Ph.D studies here at Loughborough University.

Finally, I would like to thank my parents for their continuous and endless support throughout my education. I would have not reached this stage without their patience and encouragement.

TABLES OF CONTENTS

CHAPTER 4. A HEURISTIC FOR MULTI-LEVEL LOT-SIZING PROBLEMS WITH BOTTLENECK(S)

CHAPTER 5. AN ANALYSIS OF MULTI-LEVEL LOT-SIZING PROBLEMS WITH BOTTLENECK UNDER A ROLLING SCHEDULE ENVIRONMENT

LIST OF TABLES

LIST OF FIGURES

LIST OF GRAPHS

ABSTRACT

Lot-sizing scheduling techniques determine what amount is required to meet forecasted demand whilst minimising the sum of setup and holding costs. These techniques are not adequate to provide an optimal solution to bottleneck facility problems which do not meet demands placed on them. Thus, it is necessary to analyse how much should be produced from each product with bottleneck facilities. Therefore a lot-sizing problem with bottleneck(s) under rolling schedule environment is the subject of this thesis.

This research proposes a simple heuristic for multi-level lot-sizing problems where there is a bottleneck. Previous methods to solve this problem have formulated the problem as an integer programming problem and solved the problem using a Lagrangian relaxation embedded within the branch and bound procedure. Then the proposed heuristic is extended for multiple bottleneck problems, and finally applied to the real life problem.

In this research it is suggested that items to be produced can be grouped into two types and a simple but efficient heuristic can be used to determine the production quantities required. A program was developed to compute production levels and was found to require only a small fraction of the computer time required by the full integer programming approach and to produce solutions of reasonable quality. The heuristic is simple to implement.

Keywords: heuristics, inventory, lot-sizing, scheduling, production, bottleneck, linear programming, integer programming.

CHAPTER 1

INTRODUCTION

1.1. Introduction

The aim of a production planning system is to produce one or more products to satisfy demands over the planning horizon whilst minimizing the total production costs. To be able to detect the optimal solution or near-optimal solution for production planning problems it is important to correctly specify the structure of the system and the characteristics of costs. The most successful system will usually be based on a model which gives computationally large gains from improvements to the costs or structure of the systems.

Most of the production planning models given in the literature are based on satisfying the external demand which is known in advance. It is most common to find in the literature the capacitated production planning problem where demand exceeds the system capacity. This instance may appear for different reasons: (1) incrementation of demand beyond the capacity of the system, (2) shortage of highly skilled operators, (3) scarcity of tools needed in one of the production stages. Thus, all these situations result in a bottleneck problem which does not satisfy the external demand in a manufacturing firm or in the market. Even if the firm were able to expand the capacity utilisation the new lead time required for the additional capacity utilisation would be very substantial and therefore the firm would not be able to meet the excess demand.

Organising the problem requires the simultaneous consideration of the different products, as well as the external demands on the bottleneck facilities. So, for the bottleneck facilities, the firm needs to determine how to utilise its resources. The focus will be upon how much to produce from the resources in order to minimise the total cost for the bottleneck and the non-bottleneck items.

The related carrying and setup costs for each product inside or outside the bottleneck facility are also considered, in order to minimise the related carrying and setup costs subject to bottleneck facility. This is the traditional lot-sizing and scheduling problem with bottleneck(s). This will let us consider the production scheduling and lot-sizing problems on the bottleneck facilities. The production scheduling problem is affected by the detailed level of inventory carrying costs and setup times. The traditional production planning problem model considers the setup times, and if they are negligible then they are ignored; this is a typical linear programming (LP) problem, if not the problem becomes an integer programming (IP) problem. These problems frequently result in sub-optimal solutions so that production on the bottleneck facilities becomes very important.

Recently optimised production technology (OPT) has had plenty of attention for the importance of bottleneck problems. OPT developed in 1970s as a software package, is a quantitative technique whose aim is to maximise profits by decreasing carrying inventory and expenses arising from bottlenecks. OPT adresses scheduling whilst considering the shop floor activities such as bottlenecks, lot-sizes, setups. It considers the processing times for each item allowing for bottlenecks rather than a given quantity of each item allowing for

bottlenecks. The processing time is not a part of this thesis.

However, the topic of this thesis is a heuristic for multi-level lot-sizing problems with bottleneck(s) under a rolling schedule environment.

The organisation of this thesis is as follows.

Chapter 2 reviews the relevant literature of the general lot-sizing problem, and then divides the review into: (1) the serial lot-sizing problem, and (2) the non-serial lot-sizing problem.

Chapter 3 concentrates on the material requirements planning and its deficiencies, and focuses upon the integer programming formulation for the multi-level lot-sizing problem with a single bottleneck which is adopted by Billington et al. [1986].

Chapter 4 considers a simple heuristic to solve the single bottleneck problem as an alternative to the approach of Billington et al. [1986], and extends the problem to multiple bottleneck problem cases. In this chapter, production items are divided into two types which are: (1) bottleneck items, (2) non-bottleneck items. For the bottleneck items, a heuristic is developed, and used. For the non-bottleneck items, two well known heuristics, the Economic Order Quantity and the Silver-Meal heuristic, are used. Finally, sensitivity analysis is applied to the cost structure.

Chapter 5 provides further literature review and testing for multi-level lot-sizing problems with bottleneck(s) under a rolling schedule environment.

Chapter 6 discusses the results of chapters 4 and 5, then provides a worst case analysis.

Chapter 7 provides an application of the heuristic to the assembly product structure to compare the results of this heuristic with the Eftekharzadeh [1988] approach.

Chapter 8 finishes the thesis with a conclusion and suggestions for further research.

CHAPTER 2

REVIEW OF THE LITERATURE

2.1. Introduction

This research as mentioned in the previous chapter concentrates on multi-level lot-sizing problems with bottlenecks under a rolling schedule environment and will focus on using a simple heuristic to solve problems. In this chapter, the first section explains the general lot-sizing problem, then the discussion splits into two sections which are serial lot-sizing and non-serial lot-sizing problems. The literature related to the rolling schedule will be given in chapter 5. Those subtopics given above will now be reviewed in detail.

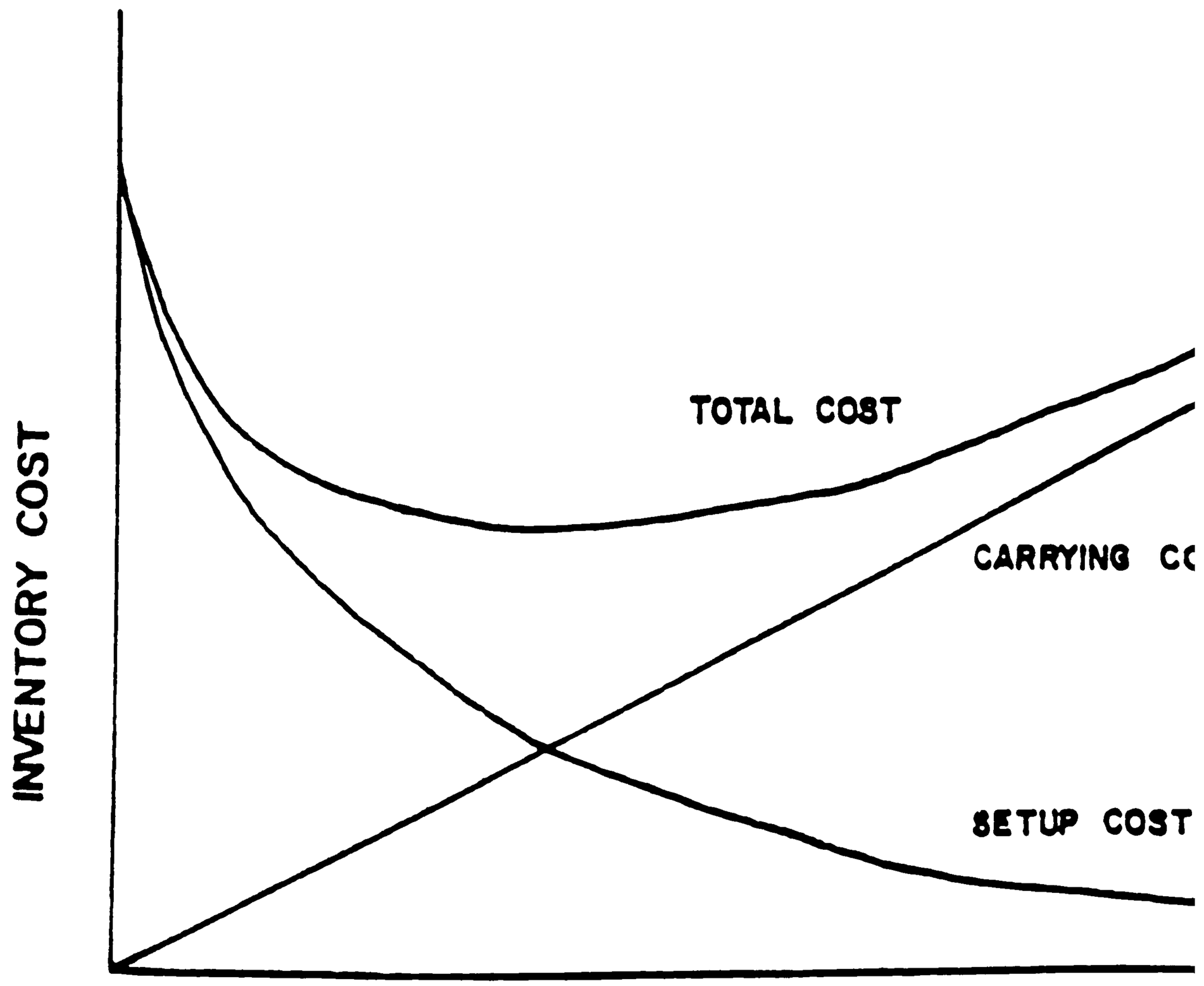
2.2. The General Lot-Sizing Problem

The main problem for many firms is to decide how much they will produce with limited resources. This is a dilemma for the production managers in selecting a procedure. The question of 'how much to produce?' refers to a lot-sizing decision which is known as a production order quantity. It combines the requirements and the production orders in the planning horizon. Lot-Sizing rules do not provide the correct period for placing the requirements but they determine the order quantities. The specific procedure for determining the quantity of orders for a part or finished product is given by the lot-sizing rules. The objective of these rules is to choose the lot-sizes which minimise the total of setup and inventory holding costs. A setup cost which is not dependent on the order quantity is incurred whenever an order is placed

during the planning horizon. Holding costs depict the cost of carrying production for some periods and are charged in accordance with the ending inventory. The sum of these two costs is referred to as the total inventory cost or total cost. The relation between these two costs is illustrated in Graph 2.2 (adapted from McLaren [1977]).

McLaren [1977] classifies the lot-sizing problems from the literature according to (i) the number of levels; (ii) the number of end items; (iii) the continuity of demand; (iv) the constraint of system capacity; (v) the availability of production systems. (i) refers to the stages in a manufacturing firm. Some authors use echelon terminology to depict this classification. The echelon inventory is defined as the whole units which have been produced at stage j. That is, cumulative production less cumulative requirements at stage j in period t and as a result of this inventory, echelon cost is the cost charged to each unit of the echelon inventory. (ii) refers to the finished items, (iii) refers to the inventories which are removed gradually from the system through the planning horizon, (iv) refers to whether the system capacity is limited or not, and finally (v) refers to the type of product structure. He then developed a setup cost adjustment procedure for single-level lot-sizing problem without capacity constraints. He showed that the Wagner-Whitin heuristic using the modified setup cost procedure provides the lower costs compared to unmodified heuristics. No adjustments were made to the holding costs at any stage.

Figure 2.2 illustrates several different types of product structures. By referring to the bill of material in a MRP (Material Requirements Planning) environment the number of predecessor items needed for successor items is established. (These numbers are not depicted in the



Graph 2.2. Total Cost versus Lot-Size.

Figure 2.2.,1,2,3,4).

Collier [1980] discussed the effect of the following lot-sizing heuristics; Lot-for-Lot (LfL) which establishes a separate lot-size for each period, Economic Order Quantity (EOQ) which always orders the economic order quantity, Part Period Order Quantity (POQ) which calculates the lot-size for the next T^* periods, where T^* is the EOQ divided by average demand and rounded to an integer value, Least Total Cost (LTC) which is based on the rationale that the sum of setup and inventory cost (total cost) for all lot-sizes within the planning horizon will be minimised if these costs are nearly equal as possible, and Wagner-Whitin (WW) which enumerates all possible ordering combinations. He showed the effectiveness of the heuristics according to the setup and inventory costs, overtime hours or work centre load profiles. His results show that the superiority of any lot-sizing rules to the others is dependent on characteristics such as the cost structure of the system

Biggs et al.[1980] examined various lot-sizing rules and procedures under different parameters, and concluded that the Lot-for-Lot (LfL) rule, which satisfies the requirements in each period, is generally very attractive in comparison to the other heuristics. This is true when the Lot-for-Lot rule is applied to systems in which the setup costs are very low and inventory costs are very high. Although it is very easy to use, because of satisfying the demands in each period it is not economical. Another lot-sizing rule, fixed order quantity, is not economical either because it places orders of the same size for each period, and is unusable for varying the quantity.

Askin and Raghavan [1983] examined three lot-sizing rules employed on work centre load variability. Lot-for-Lot (LfL), Economic Order Quantity (EOQ) and Silver-Meal rules were used to show their effect via simulation. They observed that lot-sizing rules increase the workload variation which causes production level change costs such as overtime, undertime, hiring and firing, and these costs have to be incorporated into the economic analysis.

There are many lot-sizing rules in the literature examined by researchers in detail, such as Orlicky [1975], Berry [1972], Johnson and Montgomery [1974], Monks [1987], and Browne et al. [1988]. Although a number of lot-sizing rules have been proposed, there is still little guidance for managers to choose the best rules for their system. Ramsay [1977] presented a broad classification for different lot-sizing techniques. Because of the structure of the system, these lot-sizing problems could be divided into two categories: serial and non-serial systems. The related literature in conjunction with these two systems will be given in turn in the remaining sections.

2.2.1 Serial Lot-Sizing Systems

The basic form of the multi-level lot-sizing problem, that is the most simplified form, involves producing one product in a multi-level stage uncapacitated production process, and is called a serial system. Figure 2.2.1 shows this production structure. The multi-level characteristic is derived from the hierarchical nature of the manufacturing process where raw materials are processed into components, components into subassemblies, and so on until the final product is completed.

Zangwill [1966] studied a dynamic programming lot-sizing rule for the multi-echelon serial system using uncapacitated stages, concave production cost and time varying demand. An inventory system may consist of several stocking point. In some cases these stocking points are organised such that one point acts as a supply point for others. This type of operation may be repeated at different levels so that a demand point may again become a new supply point. This situation is generally referred to as a multi-echelon systems. He developed a set of extreme point solutions in which the lots make up an integer number of periods. The set of all extreme points of all basic sets was defined as the dominant set. He then showed how to search for the optimal solution from the dominant set of production schedules with the extreme flow properties. The computation time is large, however.

Love [1972] considers the serial lot-sizing system with non-increasing production cost over time and positive echelon costs in all stages. He recommended that the lot-sizes of a component will be integer multiples of its parent's lot-sizes in a serial system, for example, if the parent's lot-size is 200, the subparts lot-size could be 200, 400, 600, etc. His approach reduces the number of lot-size calculations for the highest level part, and then finds the integer multipliers of lot-sizes for the components at lower levels. Jensen and Khan [1972] used the same integer multipliers concept with non-constant demand for serial systems.

Schwarz and Schrage [1975] examined a different approach for serial multi-level lot-sizing problems. They developed a myopic system algorithm which determines lot-sizes by taking two adjacent levels in the product structure at a time. This research assumes that production and demand rates are continuous and constant,

and the planning horizon is infinite. Thus, once the lot-size for a component is determined , it will not change any more. The major difficulty with their method is the complexity of the structure because it is based on branch and bound methodology.

Lamrecht and Vander Eecken [1978] presented a method to solve the serial structure problem with a capacity constraint which must apply to the final item. They use the Florian and Klein [1971] results to illustrate whether there are a finite number of solutions for the capacitated problem. Results show that any suitable lot-sizing heuristics could be used to test the optimality for the predecessor items, and then the same procedure is repeated for the end items within the capacitated problem. More complex structures and capacity constraints on the other items are not included in their research.

Gabbay [1979] studied a parallel problem shown in Figure 2.2.2, which consists of a set of serial structures, with capacity constraints on every level assuming setup cost and time are zero. The results illustrate that the algorithm is based on very restrictive assumptions; namely, each production item has to consume the same production time on each machine to be produced on each level. His algorithm does not provide a feasible solution for some cases according to the above restricted assumptions.

Zahorik et al. [1979] studied the parallel serial problem with constraints on total throughput (bundle constraints) assuming production costs are linear, i.e. setup costs are zero. They illustrated that the problem is a network problem when there are only three periods and when there are restrictions on the location of the bundle constraints. They used the three period results as the basis of their heuristic.

Ramsay [1980] studied the capacity constraints at each stage in a serial structure. He developed a branch and bound procedure based on the Lagrangean relaxation method. Results illustrated that the capacity relaxed formulation generally results in lower bounds than the integer linear programming solution, but the lagrangean relaxation method embedded within the branch and bound procedure does not give a feasible solution for some problems.

Maxwell and Muckstadt [1985] examined the lot-sizing problem with instantaneous production. They formulated the problem according to the reorder intervals which are the power of two multiples of a basic period rather than lot-sizes and proposed a nested schedule as one for which, if any stage produced in a given time period, then so did the next stage in serial structure.

Billington et al. [1986] considered capacity constrained multi-level parallel scheduling problems, with a bottleneck which occurs at the final items. Their formulation was based on Billington et al. [1983] except that there was one capacity constrained work centre. They eliminated inventory by substituting cumulative production minus cumulative demand rather than using both inventory (I_{it}) and production (P_{it}) variables. They solved the problem using a branch and bound procedure embedded within a Lagrangean relaxation method with a simultaneous determination of lot-sizes, lead times, and capacity utilisation plans. Although they involve setup times in the formulation, they assumed the setup times to be zero and the method provided by authors was complex. However, their approach is particularly comprehensive and it provided the inspiration for much of the work of this thesis. The details of this approach will be given in chapter 3.

Billington et al. [1988] discussed two different classes of heuristics. The first class of heuristic, which is called period-by-period, was based on the Lambrecht and Vanderveken [1979], and the Dixon and Silver [1981] heuristics. The second class of heuristic, which is called the four-step algorithm, was based on the Dogramaci et al. [1981] heuristic. The first class of heuristic proceeds on a schedule period by period starting with period 1. It determines the production lots in each period. In addition to Eisenhut [1975] their heuristic takes into account the varying demand and feedback capability. The second class of heuristic starts with a schedule using the Lot-for-Lot method for each item. This heuristic, as opposed to the first one, considers the whole period. They compared these two classes of heuristic assuming all stages are constrained by capacity and all demands are satisfied without backlogging. They showed that the Dixon and Silver [1981] heuristic was the most efficient heuristic for the single-level lot-sizing problem which involves determining the production quantities for a single part, and also two level serial systems. The reason being that it requires higher order polynomial time than the first class of heuristic.

Hum [1988] examined a two product single stage bottleneck facility in a serial system under different assumptions which are; (i) no bound on demand, (ii) lower bound on the demand, (iii) lower and upper bound on demand. The basic idea in this research was to generate some insights to the integrated mix planning, lot-sizing problem. He restricted his research first to the common cycle approach in which each product is manufactured once only in each cycle, then extended this research to allow for multiple stages.

Maes and Wassenhove [1991] considered capacitated lot-sizing problems in a parallel structure under dynamic demand conditions. They extended their research [1986] to multi-level lot-sizing problems. They investigated several conventional cost approaches introduced in Blackburn and Millen [1984] and showed that the k-branch and bound method is capacity sensitive and performs better than the other heuristics.

2.2.2. Non Serial Lot-Sizing Systems

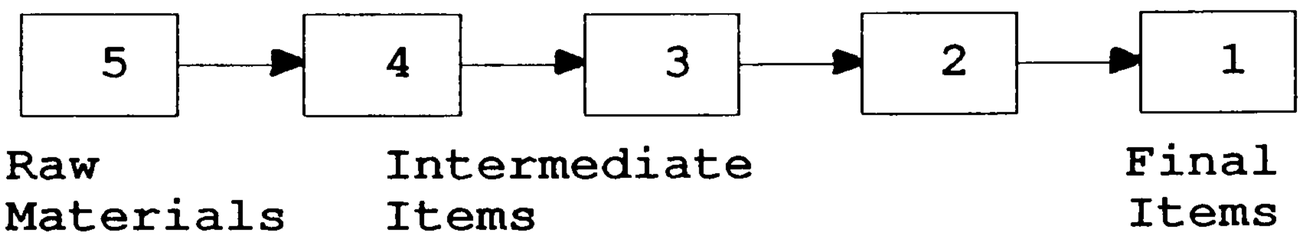
In this section two different sets of product structure literature will be reviewed. They are: Assembly product structure, which may have only one successor and more than one predecessors; and general product structure, which may have more than one successor and predecessor. Figure 2.2.3 and 2.2.4 illustrate these production structures.

Crowston et al. [1972] illustrate the integer multiple ideas which are appropriate for the assembly systems where each part may have many predecessors but only one successor. They recommend three heuristic routines and a dynamic programming algorithm to find the integer multipliers. They assumed constant demand, that is the demand at any period is assumed equal to the average of the (assumed) known demands for all periods under consideration. Crowston et al. [1973] established that optimum lot-size must be an integer multiple of its successor items at each stage. They illustrated that the time required to solve this problem increases linearly with the number of stages and exponentially with the number of time periods. Their algorithm is quite complex from the point of view of computation and capacity constraints are not included.

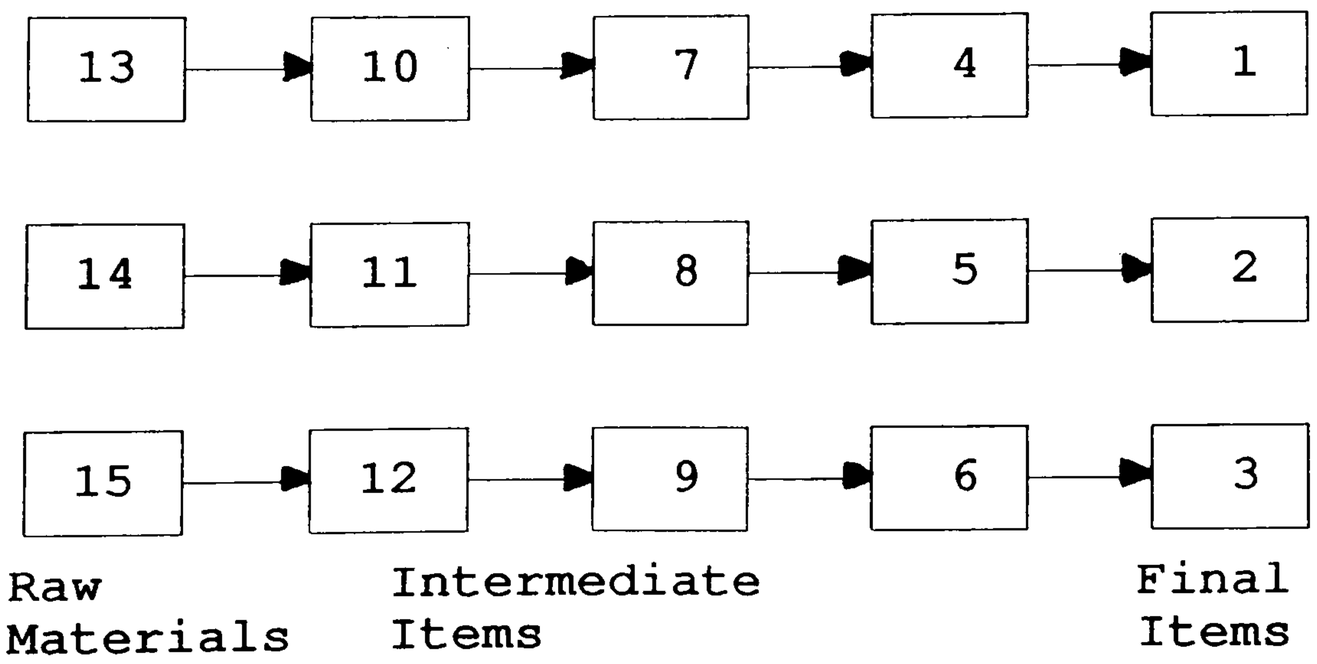
Blackburn and Millen [1979a, 1979b] tested several lot-sizing methods without capacity constraints. Blackburn and Millen [1982] classified lot-sizing methods into two main groups: (i) analytical methods which provide an optimal solution to the problem, (ii) heuristic methods which provide an optimal or near optimal solution to the problem. They then tested some of the existing lot-sizing heuristics which yield good cost performance for assembly systems. The results illustrate that if the cost information is passed from one level to successor level, the cost performance could have improved.

Figure 2.2. Different Product Types

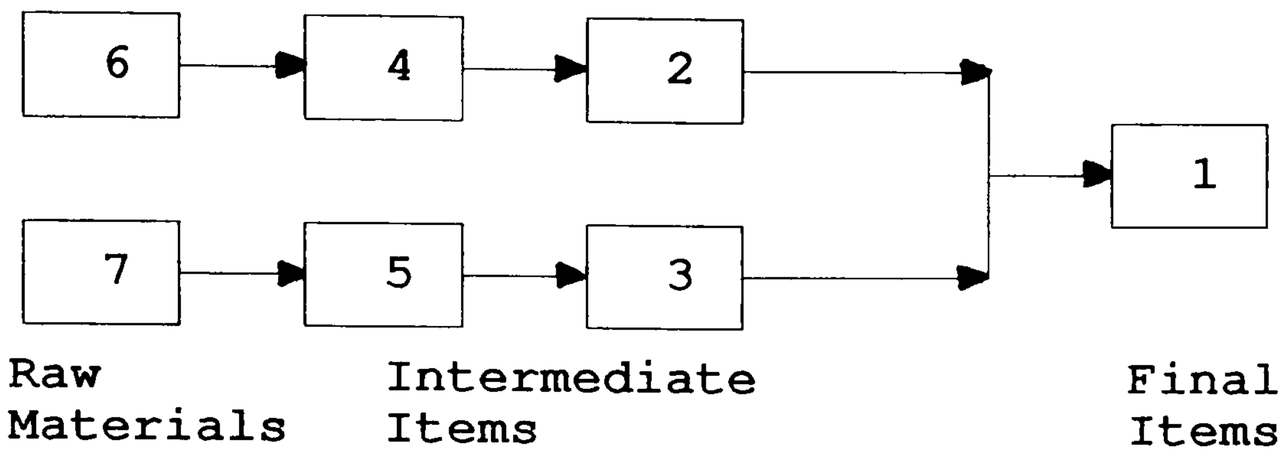
1. Serial Product Structure: Single product, produced in a series of steps.



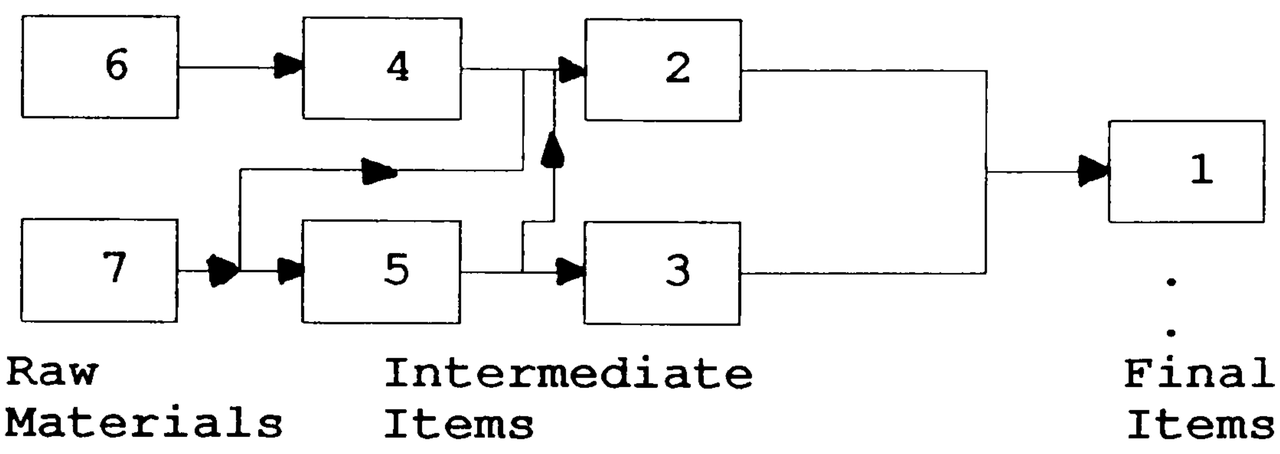
2. Parallel Product Structure: A collection of serial structures.



3. Assembly Product Structure: A product made by a complex assembly process.



4. General Product Structure: Commonality of components.



The cost performance is measured by the sum of the holding and inventory cost over the planning horizon. They tested several cost performance heuristics, but did not allow capacity constraints in their research. A later paper of Blackburn and Millen [1984] allowed the constant capacity constraints at each level to make lot-sizing decisions in a Material Requirements Planning (MRP) environment. This research is based on McLaren [1977]. They formulated the problem to minimise the sum of setup and holding costs in the following way (adapted from Blackburn and Millen [1984])

$$(P) \qquad \text{Minimise} \quad \sum_{j=1}^{N} \left[\frac{s_j}{n_j} + \frac{h_j D_j n_{p(j)} (k_j - 1)}{2} \right]$$

Subject to

$$n_j = k_j n_{p(j)} \qquad j= 2,...,N$$

$$n_j \leq V_j/D_j \qquad j= 1,...,M$$

$$n_j, k_j \geq 1 \text{ and integer}$$

Where s_j is the setup cost at stage j, h_j holding cost at stage j and V_j is the system capacity at stage j, D_j is the demand per period, p(j) is the single immediate successor stage (or parent) of stage j, n_j is the order interval for stage j where $n_{p(1)} = 1$, k_j values are the number of orders at stage j. The k_j values can be determined with different approaches. They are: (i) Unmodified Costs (UM), (ii) k-Continuous Constrained (kCC), (iii) k-Branch and Bound (kBB), and (iv) k-Branch and Bound Percent (kBB%). The first one tackles the problem with the actual setup and holding costs not attempting to taking into account any relations between stages. This approach is only used to make a comparison with other revised cost approaches. The k_j values in the kCC approach are found using differential calculus starting with the lowest stages. After finding the k_j values, the revised costs are calculated and applied to the next highest stages. The capacity constraint is ignored in this approach. The k_j values are calculated simultaneously by using branch and bound (kBB) and are used to calculate the revised costs in respect to the cost results. The capacity constraints are included independently for each item in this approach. In the k-Branch and Bound % approach, the capacity is included as a percentage of total demand. They recommended the Wagner-Whitin and k-branch and bound procedure due to their superiority to others. This is a complex approach. Blackburn and Millen [1985] compared several lot-sizing methods combining the four cost modification procedure

for single stage lot-sizing problem. They concluded that the combination method which is used with the cost modification procedures yields better results than the unmodified approaches. Their heuristics are only valid for one end item problems and the combination of these cost performance heuristics with other lot-sizing heuristics is quite complex in accordance with the computation time.

Steinberg and Napier [1980] formulated the lot-sizing problem as a network problem including commonality in the product structure. They assumed all lead times as one period. Therefore the results do not guarantee a feasible solution because the capacity constraint is not significant. They proposed a mixed integer linear programming which is applicable for small problems.

Karni and Roll [1982] propose a heuristic for the assembly structure with capacity constraints. This heuristic starts with the Wagner-Whitin algorithm which is based on searching for a lower bound in conjunction with the sum of setup and holding costs for each item whilst ignoring the capacity constraint in the first stage. Their heuristic secures the feasibility condition and improves the feasible solution until no improvement can be made. The results are compared with an exact mixed integer programming model. This heuristic is, however, not applicable for large problems.

Billington et al. [1983] presented a mathematical programming approach to capacity constrained multi-level problems. Then they introduced Product Structure Compression as a method to reduce the problem size. The same problem is solved via a Lagrangean relaxation method using the subgradient optimization technique which provides lower bounds to the problem posed by Trigeiro et al. [1989]. They involved setup time in the formulation

and showed that it was difficult to solve the problem with large setup costs.

Bahl and Ritzman [1984] proposed a heuristic procedure for multi-item lot-sizing problems with capacity constraints. Their heuristic is based on combining the Manne's [1958] formulation with the fixed order techniques. They allow the regular or overtime to alter capacity. Their heuristic only considers T cyclical production sequences. When the problem size is increased it takes substantial CPU time to solve the problem.

Afentakis et al. [1984] presented a linear transformation of the general mixed integer programming formulation, and illustrated that the lot-sizing problem can be reformulated using echelon costs. The Lagrangean and subgradient optimisation techniques are used to develop lower bounds to the optimal solution. These bounds are incurred in a special branch and bound procedure. Their method is only suitable for the assembly structure. Afentakis and Gavish [1986] extended the same problem to tackle a complex product structure with multiple end items using the same techniques. Their approach can generate tight lower bounds for the problems. They did not include the capacity constraints in either models. Their method is again complex and requires a lot of computational time. A later paper by Afentakis [1987] proposed a parallel heuristic, which is a generalisation of the single-stage Wagner-Whitin algorithm, to optimise all stages individually for assembly structure. This optimisation is done in the forward fashion. The procedure means that the one period problem is solved, then the two period problem, and so on until the last period is reached. The disadvantage of this heuristic is that when the number of periods and stages is increased, the solution to the problem becomes more complex because

there are 2^{n-1} alternative plans for each period. Afentakis does not allow for the capacity constraints.

Rosenblatt [1985] compared two replenishment problem policies which are: the Fixed Cycle and the Basic Cycle policy. The Fixed Cycle policy is solved using a dynamic approach to find the fixed intervals in the planning period. In this case optimal combinations are found by dividing the items into groups and then finding the optimal cycle time for each group, whereas the Basic Cycle policy uses a heuristic to divide the items into only two groups. This approach is usually applied in cases where the ordering cost is of the first order interaction. Although the Basic Cycle approach is very efficient computationally, it is dependent on the data set. Hence they did not derive any conclusions. These policies have been applied to the case when items have dependent demand.

McClain et al. [1985] propose a Cyclic Schedule, which is a repeating production pattern, for the multi-stage assembly systems using power-of-two multiples between stages. Such a schedule prevents the occurrence of bottleneck(s). The cycles are based on Economic Order Quantity (EOQ) analysis which is applicable for steady demands.

Thizy and Wassenhove [1985] developed a method based on Lagrangean relaxation multipliers to get the lower bounds for the multi-item capacitated lot-sizing problem. The proposed method is complex and requires substantial computer time.

Fogerty and Barringer [1988] propose a method analogous to the Wagner-Whitin and the Baringer and Fogarty [1987] heuristics. Capacity constraint is not included in their research. They use a tableau approach and include

marginal setup costs in the minor ordering costs for each product.

Eftekharzadeh [1988] proposed two new heuristics which are: the Selective Enumeration heuristic which is a modification of the multi-stage uncapacitated lot-sizing procedure; and the Modified Per Period heuristic which is the modification of the Lot-for-Lot (LfL) heuristic. The results are compared with the mixed integer programming solution. His Selective Enumeration heuristic is similar to the Karni et al. [1982] and Afentakis [1987] heuristic except that he allows for a capacity constraint in stage 1 and also his heuristic uses the shortest path method rather than the Wagner-Whitin algorithm. His algorithm takes substantial computational time and hence it is not applicable for large problems. More discussion will be given in chapter 7 to compare his heuristic to the ones which will be proposed in this thesis.

Atkins and Iyogun [1988] extended the well known Silver-Meal heuristic for multi-product dynamic lot-size problems. This heuristic provides a lower bound obtained from decomposing the problem into single item problems. A later paper (Iyogun [1991]) improved the research of Atkins and Iyogun using Silver-Meal which combines production into lots for several periods as long as average cost keeps decreasing for several periods and a Part Period Balancing heuristic which minimises the absolute differences between the setup and holding cost until the next setup.

McClain et al. [1989] present a decomposition approach to solve large scale LP problems for multi-stage production scheduling problems with capacity constraints. They did not include setup time or setup costs, therefore the problem is reasonably easy to solve. They show that the decomposition approach is faster than LP. They allow

overtime to balance the capacity usage because of excess production.

Maes et al. [1991] developed a few rounding heuristics, starting with the LP solution, for the capacitated multi-level lot-sizing problem. They reported that although the partial branch and bound procedure yields best results in comparison to the others, it requires very large CPU time.

Gupta and Keung [1990] reviewed the heuristic literature for single-stage and multi-stage lot-sizing problems, then they showed the rolling schedule effects in making the schedules. They conclude by classifying the existing heuristic literature for product structures into (i) series, (ii) assembly, (iii) arborescent, (vi) general network. Bahl et al. [1987] categorise the current literature into single-level and multi-level production problems. These categories are restricted by unconstrained and constrained production problems. They assessed these categories on the basis of the computational effort, generalization, optimality, simplicity, and testing of the proposed heuristic . On the other hand Goyal and Gunesakaran [1990] classified the multi-stage production inventory system used into (i) the system configuration; (ii) the objectives considered; (iii) the techniques used for modelling and solutions as criteria to access the model. (i) In the system configurations single-product or multi-product problems, with single or multiple machines for multi-stage systems, were reviewed. (ii) Among the objectives considered were:- the determination of the economic production quantity and start up and shut down schedules for the systems, the estimation of production efficiency,and the determination of optimal inventory levels. (iii) In the modelling and solution techniques; conventional average cost models, linear or integer programming models,

queueing models, network models, dynamic programming, models, models based on branch and bound methods, heuristic models and simulation models were reviewed. Both reviews are very broad.

2.3. Summary

In this chapter the research and literature review were organised according to the type of product structure and to the computation time, ie whether a system takes substantial computation time or not, using heuristics or methods, for capacitated and non-capacitated problems. Lot-sizing heuristics were defined whenever they were mentioned in the text rather than as sub-sections.

CHAPTER 3

BACKGROUND WORK

3.1. Introduction

This chapter will consider in great detail the work described in Billington et al. [1986]. As was mentioned in the previous chapter this paper provided a very comprehensive treatment of the interaction between production scheduling and lot-sizing and produced some definitive results. The problem formulation used by these authors will be discussed in section 3.3, but to introduce the area a review of Material Requirements Planning (MRP) work will be given first.

3.2. Material Requirements Planning

Material Requirements Planning (MRP) is a computerized information system which determines the requirements for parts and components in multi-level multi-product production planning environments. The MRP works according to the following logic: it takes a discrete production plan for a parent item from the master production system, explodes the parent item's requirements into component items and raw materials, calculates the net requirements subtracting the available inventory from the gross requirements, then calculates the lot-sizes for all items at each stage, and finally offsets the lead times according to their due dates. Hence, the MRP system controls both the material control and planning at the same time. The main reference for this field is Orlicky [1975]

MRP includes three major categories, which are: (i) Master Production Schedules (MPS), which is the input for the MRP systems, i.e. to determine the quantity and timing for each item to be produced, (ii) Bill of Material (BOM) which provides the whole information about components and end items within the hierarchical levels, which goes into that end product for the MRP, such as their sequence, their quantity in each finished item, and the work centres to be used to produce the items, (iii) Inventory Status File, which is also called the record file, which is to provide up-to-date information for each item. It involves an identification number, available quantity and procurement lead time of each item (see Adam and Ebert [1989]). The outputs of the MRP system are the order release requirements, order rescheduling, and planned orders. According to these outputs, the production planner can meet the material requirements within the capacity and lead time context.

Adam and Ebert [1989] classified the intention of MRP systems as follows: the reduction of inventory requirements, the reduction of production lead times and delivery lead times to customers, realistic delivery commitments to customers , and finally the incrementation of operating efficiency. But it fails when unpredictable lead times, work centres with limited capacity, unpredictable external demand for parts, defective items are involved in the system. MRP systems usually assume that there is no capacity constraint (see Zangwill [1966], Florian and Klein [1971], Afentakis et al. [1984], Afentakis and Gavish [1986], Blackburn and Millen [1979]) so that they proceed according to infinite loading where it is assumed any production amount is possible on each facility. Infinite loading has been studied by many authors such as Wagner and Whitin [1958], Silver and Meal [1973], Berry [1972], Fogerty and Barringer [1988] and so on. Some systems use finite

loading by loading the facility up to capacity. McClain et al. [1989] showed how to accomplish finite loading using a linear programming approach. They allowed overtime to avoid difficulties of the bottleneck but they did not include the setup cost and time.

There are two types of MRP. They are: the regenerative and the net change approaches. In the regenerative approach, the available plan must be changed whenever there is a change in the master production schedule or inventory status file (such as revised lead times). When this change occurs, the regenerative approach restarts the planning from the beginning of the period. This is controlled by short period intervals, often weekly or monthly. On the other hand, whenever there is an unexpected event the net change approach proceeds only for those parts which are affected. The regenerative approach is well suited to a stable environment, because it is checked more often. Conversely, the net change approach could need more computer access, because it is controlled by a daily basis or even more frequently.

Maxwell et al. [1983] pointed out there is no model which can solve the entire manufacturing management problem and classified the main aspects of the production problem which must be dealt with '(1) manufacturing processes with several stages of manufacture, common items, and so on, (2) dynamic lead times that depend on the state of the system, (3) capacity limitations at multiple locations, (4) uncertainty of supply and demand, and (5) allowing for setup as well as setup cost.' They also discussed the important aspects of the production planning and control process in existing literature, and proposed a comprehensive modelling framework. They do not propose detailed models, but they illustrate what models are required based on assembly product structure.

McClelland et al. [1988] discuss the effect of the MRP environment and point out that MRP is unsuitable for the real life problem in terms of variability of an end item demand, production times, production capacities and purchasing lead times. They compare the performance of production order quantity (POQ), Lot-for-Lot (LfL) and economic order quantity (EOQ) lot-sizing techniques in terms of inventory costs and customer services.

Chae [1988] noted that the available MRP literature does not give a definitive recommendation on '(1) how to develop the production schedule, (2) how to preselect each item's lot-sizing policy, and (3) how to revise the production schedule and material requirements plan, recognizing capacity and sequencing constraints' and proposed a heuristic lot-sizing/scheduling methodology of a multi-stage capacitated production system by using load families. The production plan would be more efficient if it involves the above recommendations.

Having explained the deficiencies of the MRP technique, the integer programming formulation in MRP environment, which is adopted from Billington et al. [1986], will be given in the next section.

3.3. Integer Programming Formulation of Problem

The work of Billington et al. [1986] provides a definitive treatment of types of lot-sizing problems. The consideration of these and suggestions for their solution will form the major part of this thesis. For the lot-sizing problem a bottleneck will be defined as follows. A bottleneck is a work centre which converts raw materials into finished goods through the use of resources in the manufacturing process. Therefore a machine with limited capacity, highly skilled or specialised workers, and task-specific machines or tools can all be seen to be

bottlenecks under this definition. All the resources could be classified into a bottleneck. A situation when the work centre capacity is not enough to satisfy the demands for some periods will be subject of this thesis. A general product structure with a bottleneck facility is given in Figure 3.1. (from Billington et al. [1986]). Although this figure seems to be different from the Figure 2.2.4, both figures merely show different production steps within examples of general structures. Figure 3.1 is more elaborate than Figure 2.2.4.

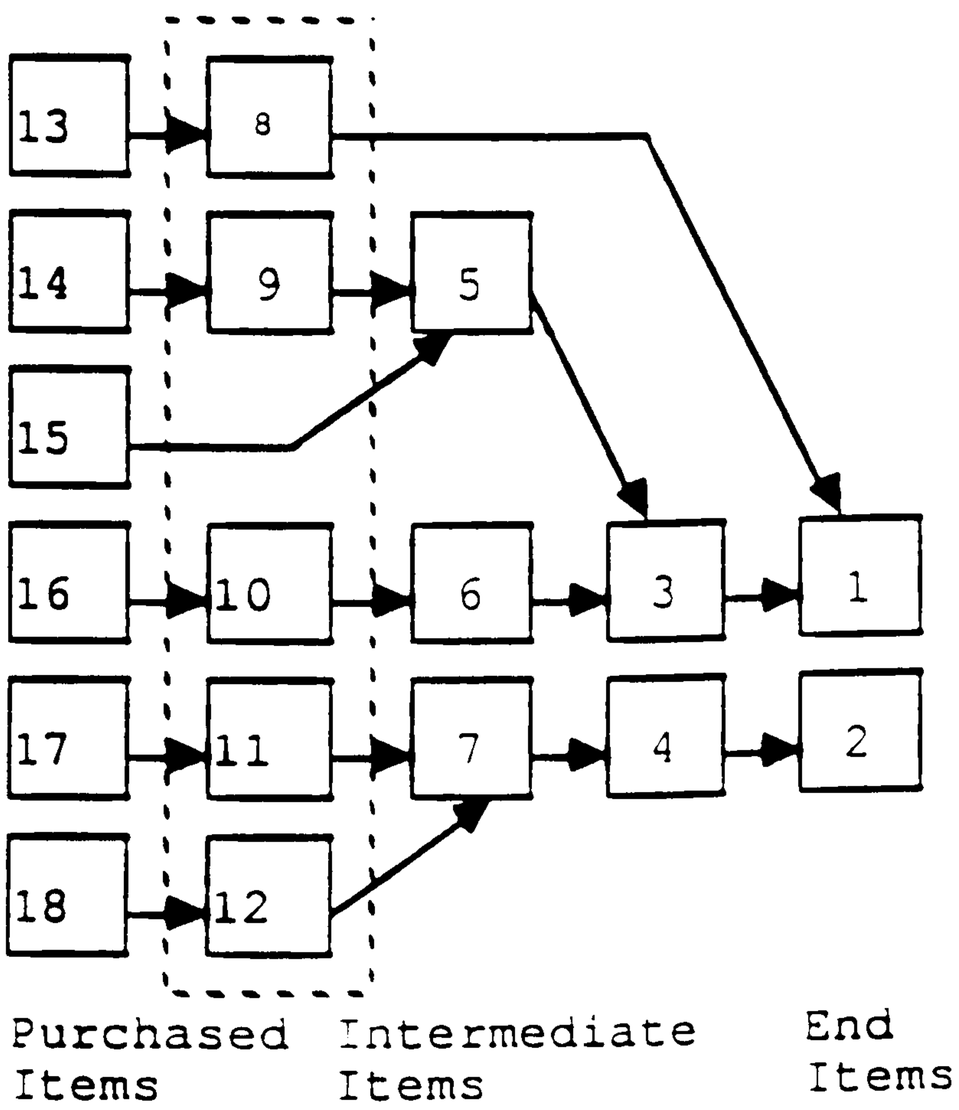


Figure 3.1. A General Product Structure with a Bottleneck Facility (The Bottleneck facility is shown by the dashed lines).

The numbers in Figure 3.1. illustrate the production items and the arrows depict the production steps. Items 13-18 are purchased items which are used to make

intermediate items, e.g. item 13 is used to make item 8, item 14 to make item 9,...,item 18 to make item 12. Items 5-7 are also intermediate items, with items 9 and 15 being required to make item 5, item 10 to make item 6, item 11 and item 12 being required to make item 7. Finally items 1 and 2 are end items, which are the final products of the process.

The general product structure, as was illustrated in Figure 2.2., can be split into a number of special cases: (1) assembly (no commonality), (2) serial (one item, multi-stage) (3) parallel (a collection of serial structures in which several items must go through the same production steps. Each item can be treated independently without a limit on resources.), (4) single-stage multi-item. This thesis will concentrate on the case of a parallel structure.

As will be seen from the numbering system in Figure 3.1, no item has a higher number than any of its predecessors. It is an a priori assumption that items in the bottleneck facility do not have predecessors (although this assumption can be relaxed for the subsequent heuristic approach). It can be seen that batching demands on product setups can result in capacity problems and also affect predecessor items since the batches are passed through as dependent demands. Because capacity utilisation varies through time, costs may not be constant.

3.3.1. Assumptions

The following assumptions provide a number of different variations for multi-level lot-sizing problems.

1. All lead times between stages are assumed to be zero,

2. Demand for the multiple end items are assumed known and at constant known rate per year,
3. There is no demand for the components at any intermediate stages,
4. Back orders are not allowed,
5. The number of units coming from bill of material required in the production of one unit at the immediate successor stage to the other stage is assumed to be equal to one,
6. Each item has only one successor and one predecessor,
7. The unit production costs are assumed constant and hence are ignored,
8. Production must occur in advance of that demand,
9. Once an item is produced, it remains in inventory for the whole planning horizon, i.e. inventory is not perishable,
10. Initial inventory is zero.

Note: These are also the assumptions used by Billington et al. [1986]. Their consequences will be discussed as appropriate later in the thesis.

3.3.2. Notations

a_{ij} = The quantity of product i needed per unit of production of product j; $a_{ij} = 0$ for all $i < j$,

b_i = Time needed on the bottleneck facility for the production of product i, and $b_i = 0$ if there is no production on the bottleneck facility,

cap_t = Units of time available at the work centre,

cs_i = Setup cost for product i (following the assumption made by Billington et al. [1986], for consistency, the possibility of " carrying over " a setup from one period to another is not allowed),

d_{it} = External (independent) demand for product i during time t, and there is no demand for the intermediate items,

h_i = Holding cost for product i,

I_{it} = Final inventory level of product i in period t,

l_i = Lead time, which is the unavoidable time from the time the order placed until it is available, for product i. This could be because of the time taken by a vendor to deliver a product, or could be a non production lag,

N = Number of products,

P_{it} = Units of i produced in period t,

S_i = Setup time for the work centre for product i. This takes the value zero for all items except those made on the work centre. This can also include processing time which is not related to the size of batch as in some heating operations,

T = Total number of periods,

X_{it} = Production indicator; equal to 1 if $P_{it} > 0$ and zero otherwise.

3.3.3. Mathematical Model

The aim of this model is to provide the lot-sizes which minimise the total inventory and setup cost along the planning horizon for all parts in the product structure when there is a bottleneck(s). The lot-sizes are constrained to meet demand for each period at least. In the formulation below inventory is eliminated by substituting cumulative production minus cumulative demand, and it is derived from a model in Billington et al. [1983].

Formulation

Minimise:

$$Z = \sum_{i=1}^{N} \sum_{t=1}^{T} h_i \ (T - t + 1) \ P_{it} + cs_i \ X_{it}$$

In this equation, production cost, which decreases linearly with time, is included as an inventory cost. The objective function in an integer programming problem generally includes the labour cost but in our case it is ignored because the labour cost is fixed. In general, the setup time and setup cost are very critical factors in production scheduling problems. When they drop from the formulation, the problem becomes a linear programming relaxation problem and is easily solved, otherwise integer programming may not give the feasible solution for some problems and takes substantial CPU time. The idea for this objective function is to find a tradeoff between the holding and setup cost which minimises the total cost. Sequencing to produce items is not included in this model.

Subject to:

$$\sum_{n=1}^{t} [\ P_{i,n-1i} - \sum_{j=1}^{t} a_{ij} \ P_{jn} \ ] \geq \sum_{n=1}^{t} d_{in} - I_{io} \quad \begin{matrix} i=1,\ldots,N \\ t=1,\ldots,T \end{matrix}$$

This constraint illustrates that available production after subtracting the requirements is greater than or equal to the external demand by eliminating the inventory in the planning horizon. This equation shows that the production must be available at least $n-l_i$, where l_i is the lead time, time periods before required. The second

summation before the inequality illustrates the interrelation between the successor and predecessor items (sometimes called father-child relations).

$$\sum_{i=1}^{N} [\ b_i P_{it} + s_i X_{it}\ ] \leq cap_t \qquad t=1,\ldots,T$$

This constraint is the capacity constraint for the bottleneck facility. This equation depicts the restricted case where the capacity is exceeded by demand for a particular item in the current period.

$$X_{it} = \begin{cases} 1 & \text{if } P_{it} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad i=1,\ldots,N$$

This constraint shows setup cost and time are applicable only if there is a production. The non-negativity constraint is illustrated below.

$$P_{it} \geq 0 \qquad i=1,\ldots,N \text{ and } t=1,\ldots,T$$

The above formulation is the integer programming formulation when there is one capacity constrained work centre.

Billington et al. [1986] present a price-directed approach, which is based on Lagrangean relaxation embedded within a branch and bound procedure, for the multi-level lot-sizing problem with a bottleneck. The solution method in their research is the branch and bound method with heuristics. They propose two solution phases which are dual and primal procedures. They define a

subproblem by assigning a fixed value to some X_{it} values at any node in the branch. They solve this problem using a Lagrangean heuristic, which relaxes all the capacity and all inventory balanced constraints for the production lot-sizes and a smoothing method is used to adjust the production in a primal phase. They report that this procedure yields a schedule which is difficult to obtain in a MRP environment. Then their procedure extends the primal phase to a dual phase which yields modified setup costs and production in each period. The aim of these two phases is to discourage production on the bottleneck facility and also improve the capability of decision making of level-by-level lot-sizing routines. The primal phase is repeated with the modified prices, which are found in the dual phase, until a good solution is reached. Subgradient optimization and a heuristic which is based on duality theory are used to force production decisions to more closly satisfy the constraints in the dual phase. This heuristic which is mentioned in Billington et al. [1986] is complex and will not be dealt with in this thesis.

In particular Billington et al. [1986] concentrated on the three problem types: 1-end-item, 3-end-item, and 5-end-items and restricted these items so that they were the only ones affected by the bottleneck. In the computational testing N has maximum value 25 and T has maximum value 12, the values chosen in the work of Billington et al. [1986].

3.4. Summary

In this chapter the MRP environment and its deficiencies was discussed. The integer programming formulation which was adopted by Billington et al. [1986] was also illustrated. Finally the derivation of Economic Order Quantity from the integer programming formulation to use

for the non-bottleneck items in the next chapter will be depicted.

Appendix

Derivation of EOQ

Essentially Billington et al. [1986] uses the objective of total cost for each product (i).

Let the total cost for item i be TC_i

$$TC_i = \sum_{t=1}^{T} cs_i \, X_{it} + \sum_{t=1}^{T} P_{it}(T - t + 1) \, h_i$$

Assume that total demand = total supply

therefore $\sum_t P_{it} = \sum_t D_{it}$ for each i (= D, say)

Let Q_i be EOQ of product i (use Q from now on)

$$\text{Number of setups} = \frac{D}{Q}$$

$$\text{therefore total setup cost} = \frac{Dcs_i}{Q}$$

$$\text{time between setups} = \frac{QT}{D}$$

therefore holding cost

$$= QTh_i + Q\left(T - \frac{QT}{D}\right)h_i + Q\left(T - \frac{2QT}{D}\right)h_i +$$

$$Q\left(T - \frac{3QT}{D}\right)h_i + \ldots + Q\left(T - \frac{(D-Q)T}{D}\right)h_i$$

$$= QTh_i\left[1 + \left(1 - \frac{Q}{D}\right) + \left(1 - \frac{2Q}{D}\right) + \left(1 - \frac{3Q}{D}\right) + \ldots + \left(1 - \frac{D-Q}{D}\right)\right]$$

$$= QTh_i\left[1 + 1 + \ldots - \frac{Q}{D} - \frac{2Q}{D} - \frac{3Q}{D} - \ldots - \left(\frac{D-Q}{D}\right)\right]$$

In this parenthesis there are D/Q terms in the first section and (D / Q-1) terms in the second section. The second section is an arithmetic progression. As the sum of (1 + 2 + 3...+ n) is equal to

$$S = \frac{n}{2}(2a + (n - 1)\ d)$$

where a is the first term, d is the common difference, and n is the number of terms, then the expression in parenthesis is

$$S = \frac{1}{2}\left(\frac{D}{Q} - 1\right)\left\{\frac{2}{D} + \left(\frac{D}{Q} - 2\right)\frac{Q}{D}\right\}$$

$$S = \frac{1}{2}\left(\frac{D}{Q} - 1\right)$$

therefore holding cost

$$= QTh_i \left[\frac{D}{Q} - \frac{1}{2}\left(\frac{D}{Q} - 1\right)\right]$$

Total cost of setup and holding = TC

$$= \frac{Dcs_i}{Q} + QTh_i \left[\frac{D}{Q} - \frac{1}{2}\left\{\frac{D}{Q} - 1\right\}\right]$$

Differentiating with respect to Q;

$$\frac{dTC}{dQ} = - \frac{Dcs_i}{Q^2} + \frac{Th_i}{2}$$

Equating this statement to zero to obtain the minimum cost gives:

$$\text{Minimum cost batch size} = Q^* = \sqrt{\frac{2Dcs_i}{Th_i}}$$

but the average demand per period is $\frac{D}{T}$

$$\text{therefore } Q^* = \sqrt{\frac{2dcs_i}{h_i}}$$

where d is average demand per period.

CHAPTER 4

A HEURISTIC FOR MULTI-LEVEL LOT-SIZING PROBLEMS WITH BOTTLENECK(S)

4.1. Purpose of Thesis

The intention of this thesis is to:

A. Propose and evaluate a simple heuristic for the model of section 3.3 (multi-level lot-sizing problem with bottleneck) as an alternative to the approach of Billington et al. [1986]. This will form chapter 4.

B. To show how the model of section 3.3 can be extended to incorporate multiple bottlenecks. This will be included in chapter 4.

C. To show how the model of section 3.3 can be used on a rolling schedule environment. This will form chapter 5.

D. Provide the results and discussions of chapters 4 and 5. This will form chapter 6.

E. To show how the heuristic can be applied to the assembly structure. This will form chapter 7.

F. Finally, the conclusion and suggestions for future work will form chapter 8.

4.2. Introduction

Previous methods to solve the multi-level lot-sizing problem where there is a bottleneck have formulated the problem as an integer programming problem and solved the problem using Lagrangean relaxation embedded within the branch and bound procedure. In this chapter, which is summarised in Toklu and Wilson [1991], Toklu and Wilson [1991] and Toklu and Wilson [1992] (included in Appendix E), it is suggested that items to be produced should be grouped into two types and a simple but efficient heuristic can then be used to determine the production quantities required in each case. This will form section 4.4 but first the problem area will be explained in section 4.3. Then this heuristic will be extended in section 4.5 for the multiple bottleneck cases, and sensitivity analysis will be investigated in section 4.6. Finally the conclusion will be given in section 4.7.

4.3. Problem Area

The primary aim of production planning models in the literature was dependent on satisfying the external demands. Lot-sizing problems become more realistic if they involve capacity constraints and if they are incorporated in multi-level systems. Capacitated lot-sizing problems in the literature (see Billington et al. [1988], Florian and Klein [1971], Garey and Johnson [1979]) are known to be NP-hard problems which means that a given problem cannot be solved in a polynomial time. The problem is hard because optimal solution techniques are unable to solve the problems within reasonable computation times. Garey and Johnson [1979] report '...In its broadest sense, the notion of efficiency involves all the various computing resources needed for executing an algorithm. However, by the most efficient algorithm one normally means the fastest. Since time requirements are

often a dominant factor determining whether or not a particular algorithm is efficient enough to be useful in practice...'. As a result of computation difficulties, the effort in this chapter is directed to use effective heuristic approaches to those capacitated problems. Available capacity sometimes may not be enough to meet the external demands because of the shortage of tools, shortage of skilled operators or more demand than the system capacity. Thus, all these instances may cause a bottleneck problem which is the most awkward part in the manufacturing environment.

The product structure with a single bottleneck facility for the five-end-item problem is illustrated in Figure 5.3. As will be seen from the product structure there is no commonality between items and each predecessor has to be the input to one successor stage. This product structure is the special case of the general product structure and is called a Parallel Product Structure.

4.4. Simple Heuristic for Multi-Level Lot-Sizing Problem with a Bottleneck

Lot-Sizing in MRP only becomes realistic when features such as capacity constraints and the fact that systems are multi-level can be incorporated into the model. Blackburn and Millen [1982] review and add to contributions made to this area. Their work provides for simultaneous lot-sizing and capacity requirements planning in an MRP framework. However, one of the most succesful attempts to tackle the multilevel lot-sizing problem with a bottleneck constraint has been by Billington et al. [1986]. This chapter will propose a simple heuristic approach to solve the problem modelled by Billington et al. [1986] and show that if the items for production are categorised into

(a) end-items, constrained by the bottleneck

(b) non-end-items, unconstrained

then two simple procedures can be used independently, one for each category of product item, to determine the production levels of each product item. The reason for this categorisation into two groups will be explained later in this section. Solutions will be sub-optimal but of adequate quality and are easy to obtain. The method to be proposed requires only a fraction of the computation required for solution of the integer programming formulation of the lot-sizing problem. In addition the heuristic is easy to implement and program when compared to the Lagrangean heuristic approach of Billington et al. [1986] and should require much less computer time and have more practical appeal in a realistic setting.

The heuristic operates by first dividing production items into end-items and non-end-items. The reason for this is that production of each non-end item is unconstrained and so has neither any effect on the production of any other non-end-item nor on the production of the product items which are constrained by the bottleneck. As demand required of all product items is known in advance, the production decision for each non-end-item becomes a relatively simple one of when to produce in order to minimise the contribution to costs (from holding and setup costs) of each non-end-item. The fact that demand for end-items determines the demand for intermediate product items does not invalidate the independence of production of each product item as demand for end-items is known several periods in advance. The extension to dependent product item cases for different product structures is an area for further research. The problem of determining when to produce end-items is more complex as these product items must share the resources of the bottleneck. Thus for these product items the production

problem is a constrained problem. However, in general these product items are in the minority.

Define S_{it} as stock of product i at start of period t

$$\text{then} \quad S_{it} = \sum_{n=1}^{t-1} (P_{in} - d_{in}).$$

4.4.1 Non-end-items

For these product items the EOQ and Silver-Meal approaches will be used. These approaches were chosen as they are comparatively simple to operate and in general will produce solutions of good quality.

4.4.1.1. The Economic Order Quantity Approach (EOQ)

Let Q_i be the EOQ for product item i, based on setup cost cs_i and holding cost h_i. Then the following strategies are considered:

(a) Produce Q_i in period 1 and then next produce Q_i in the period when stocks would become negative if no production were made (ie find the next smallest t such that $S_{it} < d_{it}$)

Let t_i be the number of occasions on which product item i will be produced. Then

$$t_i = [\sum_{n=1}^{T} d_{it} / Q_i + 0.5]$$

and production is made in any period n whenever $S_{in} < d_{in}$.

Note that [ ] is the integer part function.

If in any period n

$$Q_i \geq \sum_{t=n}^{T} d_{it} - S_{in} \quad \text{then set } Q_i = \sum_{t=n}^{T} d_{it} - S_{in}$$

(b) Let Z_i be the quantity of product item i produced in period 1. The same quantity is next produced whenever stocks would become negative if no production were made (ie when $S_{in} < d_{in}$).

$$Z_i = \left(\sum_{t=1}^{T} d_{it} \right) / t_i$$

Continue this process through all the periods.

(c) Produce all product items in period 1.

Strategies (a), (b), and (c) are evaluated to see which leads to the smallest total inventory cost over the N periods and then that strategy is chosen.

4.4.1.2 The Silver-Meal Approach (SM)

This approach selects the lot-size which covers the number of periods to minimise the total cost per period. The average total cost function of this heuristic is given as:

$$\text{Average total Cost} = \{cs_i + h_i \sum_{t=1}^{T} (t - 1)\, d_{it}\} / t$$

and chooses the smallest period t whenever the average total cost is greater than the immediate previous period's cost (ie the average total cost of (t) > the average total cost of (t - 1)). Then the lot-size equals;

$$Q_i = \sum_{t=1}^{N} d_{it}$$

where i is the product item's number. This heuristic is chosen in this chapter in comparison with the EOQ heuristic for the non-end-items because it is very effective for varying demand cases.

4.4.2 End-items

For these product items a simple heuristic was adopted which would adopt a greedy approach to production by having few setups, but with heavy utilization of the resultant production capacity. In addition, the heuristic would operate in a cyclic manner, moving between product items or sets of product items in turn to produce reasonably smooth production. The approach has broad similarities with the work of Mclain and Trigerio [1985] except that by excluding setup time and cost they handle a problem that is easier to solve. Bahl and Ritzman [1984] also adopt a cyclic approach but do so by examining permutation schedules.

The heuristic will be described with reference to three cases.

Case (a) 1-end-item problems

Produce as much of end-item i as cap_t will allow in period 1, i.e. set $P_{i1} = cap_1$, then next produce product item i when stocks would become negative if no production were made, i.e. find the next smallest t for which $S_{it} < d_{it}$. Continue the process of producing in each period t which has this property.

If P_{in} would exceed $\sum_{t=n}^{T} d_{it}$ for any period n

then set $P_{in} = \sum_{t=n}^{T} d_{it}$.

Case (b) 3-end-item problems

A three period cycle is adopted.

Period 1 Set $P_{21} = d_{21}$, $P_{31} = d_{31}$ and $P_{11} = cap_1 - d_{21} - d_{31}$.

Period 2 Set $P_{32} = d_{32}$, $P_{12} = 0$ and $P_{22} = cap_2 - d_{32}$ provided $S_{12} > d_{12}$.
Otherwise set $P_{12} = d_{12} + d_{13} - S_{12}$,
$P_{32} = d_{32}$ and $P_{22} = cap_2 - P_{12} - P_{32}$.

Period 3 Set $P_{33} = cap_3$ and $P_{13} = P_{23} = 0$ provided $S_{13} > d_{13}$ and $S_{23} > d_{23}$. Otherwise set $P_{13} = d_{13} - S_{13}$, $P_{23} = d_{23} + d_{24} - S_{23}$ and $P_{33} = cap_3 - P_{13} - P_{23}$.

Period 4 Set $P_{14} = cap_4$ provided $S_{24} > d_{24}$ and $S_{34} > d_{34}$.

Period 5 Set $P_{25} = cap_5$ provided $S_{15} > d_{15}$ and $S_{35} > d_{35}$.

Period 6 Set $P_{36} = cap_6$ provided $S_{16} > d_{16}$ and $S_{26} > d_{26}$.

Again, if stocks of any product item would become negative, produce sufficient of that product item to satisfy demand over the next one or more periods until that product item moves into the dominant production position. The product in the dominant production position is the product for which as much as possible should be

produced after ensuring stocks of other products do not become negative. In the above heuristic product 1 is in the dominant production position in period 1, product 2 is in the dominant production position in period 2 and product 3 is in the dominant production position in period 3. Whenever the stocks of any product would become negative, produce as much of that product item to satisfy the demands according to their priority.

Continue the process in the same cyclic manner for the remaining periods. If at any stage stocks of all products are sufficient for production to be zero in any period, no production is made in that period and the cycle for the appropriate product is delayed by one period.

The priority allocation using simply the order of the data as used by Billington et al.[1986] was chosen as a basis for the thesis results. Although there is no reason to believe that this priority allocation is better than any of the alternative priority allocations, there is no reason to believe that the results would by changed or improved substantially by using an alternative priority allocation. Intensive testing of other priority allocations was performed using criteria such as

(a) Largest total demand first
(b) Most variable demand first.

Testing revealed only small differences in the results compared to the results presented in Tables 4.6.1-4.6.21 which use simply the order of the data. Total costs were neither consistently increased nor decreased by considering priority allocations determined by (a) or (b). However, a further examination of the performance of the alternative priority allocations would provide an interesting comparison for further research.

Note: on the sample data on which the heuristic was tried, despite this data incorporating some highly variable demand levels and production capacities, it was found that none of the 'otherwise' type conditions listed above ever applied.

Case (c) 5-end-item problems

In order to keep the heuristic simple, more complex cases are now treated more in the style of Case (b). The set of 5 products is simplified by considering products in just two sets rather than as 5 individual products. Here products {1,2,3} are considered a set as are {4,5}. The reason why the same priority allocation, as used in the three-end-item problems, was not applied is that the capacity was not enough to satisfy the demands for some periods such as structure 3 with 95% capacity utilisation in the five-end-item problem (this is a contradiction of the "no stockouts" assumptions). The same alternative priority allocations, which were explained in the three-end-item problem, were tested intensively in the experimental studies, and it was found that there were no significant cost differences between any of the other priority allocations.

Let $P_{1t} = P_{1t} + P_{2t} + P_{3t}$, and $P_{4t} = P_{4t} + P_{5t}$. Then the two period cycle is adopted.

Period 1 Set $P_{41} = d_{41} + d_{51}$, and $P_{11} = cap_1 - P_{41}$.

Period 2 Set $P_{12} = 0$ and $P_{42} = cap_2$ provided $S_{12} > d_{12}$
Otherwise set $P_{12} = d_{12} + d_{13} - S_{42}$, $P_{42} = cap_2 - P_{12}$

Period 3 Set $P_{13} = cap_3$ provided $S_{43} > d_{43}$

Period 4 Set $P_{44} = cap_4$ provided $S_{14} > d_{14}$

Continue this process in the same procedure for the rest of the periods in the same cyclical manner. If stocks are sufficient for product in any stage, production will be zero i.e. there is no production.

The two sets are now treated as single products and the cyclical approach of case (b), modified to a two-period cycle, is followed with the modification that when production of a product set can be larger than demand in that period, the production quantity is equal for each product in the set.

Beside, the setup cost and the holding cost for the five-end-item problem will be taken to be the average of the combined items' setup and holding costs. Total inventory cost is now the total of individual inventory costs arising from the end-items and the non-end items.

4.5. A Heuristic for a Multi-Level Lot-Sizing Problem with Multiple Bottleneck

The integer programming formulation for the multi-level lot-sizing problem was given in section 3.3.3 for the single bottleneck cases. The integer programming formulation for the multiple bottleneck would be:

Minimise:

$$Z = \sum_{i=1}^{N} \sum_{t=1}^{T} [\, h_i \, (T - t + 1) \, P_{it} + cs_i \, X_{it} \,]$$

Subject to:

$$\sum_{n=1}^{t} [\, P_{i,n-1i} - \sum_{j=1}^{t} a_{ij} \, P_{jn} \,] \geq \sum_{n=1}^{t} d_{in} - I_{io} \quad \begin{matrix} i=1,\ldots,N \\ t=1,\ldots,T \end{matrix}$$

$$\sum_{i=1}^{N} [\, b_{il} P_{it} + s_{il} X_{it} \,] \leq cap_t \qquad \begin{matrix} t=1,\ldots,T \\ l=1,\ldots,M \end{matrix}$$

$$X_{it} = \begin{cases} 1 & \text{if } P_{it} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad i=1,\ldots,N$$

$$P_{it} \geq 0 \quad i=1,\ldots,N \text{ and } t=1,\ldots,T$$

where b_{il} is the time required on the bottleneck facility l for the production of product item i, and s_{il} is the setup time for the work centre l for product item i. The rest of the notations are the same as explained in chapter 3.

The objective of the integer programming problem was to minimise the total cost whilst producing one or more product(s) to satisfy demand(s) over the planning periods.

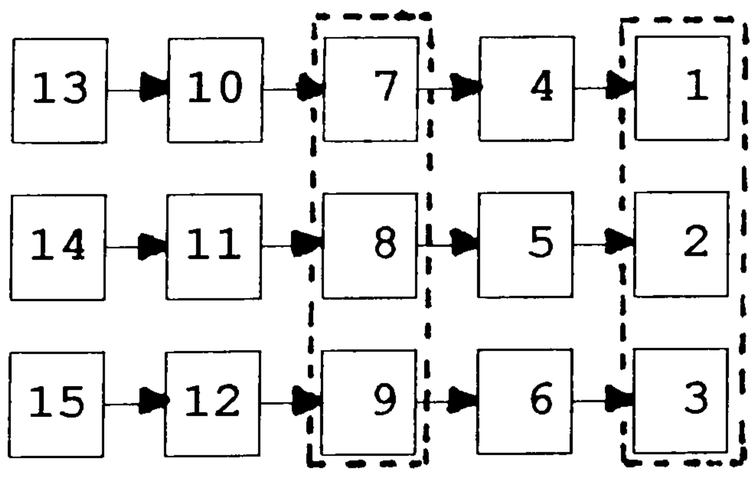


Figure 4.4. A parallel product structure with multiple bottleneck facilities (the bottlenecks are shown by the dashed lines).

The product structure for the three-end-item problem when there are multiple bottleneck facilities is depicted in Figure 4.4. Three parallel production lines are illustrated horizontally and these end on the product items, 1, 2, 3, and also the five stages are depicted vertically in Figure 4.4.

As was mentioned earlier, the bottleneck could arise when the work centre capacity is less than requirements. It is assumed that there are bottlenecks, which cause difficulty in satisfying the requirements in some periods. In Figure 4.4 the bottlenecks lie in the centre column and right hand column. The proposed heuristic for the end-items which are constrained is the same as explained in section 4.4, and the heuristic for the constrained non-end-items is as follows (and is the same as that discussed in section 4.4):

Period 1 Set $P_{81} = d_{81}$, $P_{91} = d_{91}$ and $P_{71} = cap_1 - d_{81} - d_{91}$.

Period 2 Set $P_{92} = d_{92}$, $P_{72} = 0$ and $P_{82} = cap_2 - d_{92}$ provided $S_{72} > d_{72}$.
Otherwise set $P_{72} = d_{72} + d_{73} - S_{72}$,
$P_{92} = d_{92}$ and $P_{82} = cap_2 - P_{72} - P_{92}$.

Period 3 Set $P_{93} = cap_3$ and $P_{73} = P_{83} = 0$ provided $S_{73} > d_{73}$ and $S_{83} > d_{83}$. Otherwise set $P_{73} = d_{73} - S_{73}$, $P_{83} = d_{83} + d_{84} - S_{83}$ and $P_{93} = cap_3 - P_{73} - P_{83}$.

Period 4 Set $P_{74} = cap_4$ provided $S_{84} > d_{84}$ and $S_{94} > d_{94}$.

Period 5 Set $P_{85} = cap_5$ provided $S_{75} > d_{75}$ and $S_{95} > d_{95}$.

Period 6 Set $P_{96} = cap_6$ provided $S_{76} > d_{76}$ and $S_{86} > d_{86}$.

Whenever stocks of any product item would become negative, then produce as much of that product item for the next one or more periods to satisfy the demands.

Different alternative priority allocations were tried to test the heuristic in the experimental studies, but the above heuristic using the chosen dominant production positions (see section 4.4.2, case (b)) were applied.

For the unconstrained non-end-item problems, the EOQ and Silver-Meal approaches are applied. The total cost is the sum of the individual costs.

The proposed heuristic for the one- and five-end-item problems with multiple bottlenecks was the same as was explained in section 4.4 for the end-item(s). It is used for the constrained non-end-items with only one difference, the numbers are changed. For example for the one-end-item problem, production of product item 1 is replaced by production of product item 3 where the second bottleneck is located. Similarly for the five-end-item problem, production of product items 1, 2, 3, 4, 5 is replaced by production of product items 11, 12, 13, 14, 15.

4.6. Sensitivity Analysis

In this section, sensitivity analysis is explained in an example problem for the one-end-item problem. The data used in Billington et al. [1986] was the demands for 12 periods (33, 43, 36, 46, 46, 42, 38, 41, 44, 40, 35, 46), holding costs (0.5, 0.1, 0.5, 0.1, 1.0), and setup cost (300, 200, 200, 500, 400) for the different product items. The same data is used to solve the problem by reducing the setup and holding costs by 20% and 40% respectively. The results are illustrated in Table 4.6.

Plossl and Obec [1967], and McClain and Thomas [1980] reported that 'adjusting all Q^* (Economic Order Quantity) values upward until n (number of orders) drops to its original value of 23.5 per year for the sample implies 82.1 orders per year, which is 3.49 times larger than the desired value of 23.5.' McClain and Thomas [1980] showed that by multiplying all Q^* (Economic Order Quantity) by 3.49 in their data set, a reduction of around 50% in the total cost would be achieved with no change in the annual ordering load, simply by ordering in proportion to the EOQ. This calculation was dependent on the data sets. Once the data set is changed, it is difficult to get the same reduction. In this section reductions by 20% and 40%, respectively in the original holding and setup costs were considered rather than changing the order quantity. The results in Table 4.6 showed that any reduction or increase in the original setup and holding costs, using either the Economic Order Quantity or the Silver-Meal approaches for the unconstrained product items with the proposed heuristic for the constrained (bottleneck) product items, give rise to a proportional reduction/increase in the total cost.

Because of the assumptions in the lot-sizing heuristics of uncertainty in demands, varying demands, varying costs and unpredictable demands, it is worthwhile studying the robustness property of the heuristic over the planning horizon.

Table 4.6. Sensitivity analysis for one-end-item problem

Item	Projected total cost Using EOQ			Projected total cost Using Silver-Meal		
	original	20% re.	40% re.	original	20% re.	40% re.
1	3520.0	2816.0	2112.0	3520.0	2816.0	2112.0
2	788.0	630.4	472.8	937.4	749.9	562.4
3	2558.0	2046.4	1534.8	2546.0	2036.8	1527.6
4	1088.0	870.4	652.8	1088.0	870.4	652.8
5	5116.0	4092.8	3069.6	5092.0	4073.6	3055.2

4.7. Summary and Conclusion

The integer programming formulation for single- and multiple bottleneck problems was solved by means of software MGG [1987] and SCICONIC [1986] (discussed in Appendix A), and the heuristic approach programmed in Fortran 77 (included in Appendix C) as an alternative to the integer programming. The data of Billington et al. [1986] was used to test these problems. The heuristic and integer programming results for multi-level lot-sizing problems with a bottleneck are illustrated in Tables 4.6.1-4.6.9 as well as the results of the reduction of the original costs by 20% and 40% respectively. In conjunction with these results the graphs are given in Graph 4.6.1-4.6.9. These graphs are organised to compare the results according to the linear programming (LP), integer programming solution (IP), heuristic solution for the constrained product items with the Economic Order Quantity (H1) and Silver-Meal (H2) approaches for the unconstrained product items. In these graphs 50%, 75%, and 95% of capacity utilisation are illustrated on the horizontal axis, and current values, CPU time and optimality are depicted on the vertical axis. In

addition, the results in Tables 4.6.10-4.6.12 show the total costs when each of holding costs and setup costs are reduced by 20% in turn. Different capacity utilisations such as 60% and 80%, were used to test the heuristic and the results are shown in Tables 4.6.13-4.6.18. The results in Tables 4.6.19-4.6.21 show the multiple bottleneck problem with the integer programming solution and the heuristic solutions. The graphs for the multiple bottleneck solutions are illustrated in Graph 4.6.10-4.6.12. On the horizontal axis, numbers are used to show the utilisation. Number 1, on the horizontal axis, shows the 50% utilisation in product item 1 and 25% in product item 3 for the one-end-item problems. The number 2 depicts 75% and 50% utilisation according to the bottleneck 1 and 2 which are located at product items 1 and 3. Finally the number 3 illustrates the bottleneck utilisation of 95% and 75% in relation with the product items 1 and 3 respectively in the one-end-item problem product structure. In each table, linear programming solutions (LP) are given in the third column, integer programming solutions (IP) using the branch and bound procedure are given in the fourth column, and the heuristic solution for the constrained product items with the Economic Order Quantity (EOQ) and the Silver-Meal approaches for the unconstrained product items are given in the fifth and sixth columns respectively.

According to the results, the heuristic produced a lower cost than the integer programming solutions. The main contribution in this thesis is to show that the required computer times of the heuristic to solve the problems takes a few seconds (all less than or equal to 3.31 CPU seconds). The results illustrated that only a small amount of computer time is required to solve the problems by means of the heuristic. The results also showed that reducing the original cost by 20% and 40% decreased the total costs by that percentage, but it did not yield the

same reduction when each of the original holding and setup costs were reduced by 20%. The multiple bottleneck results imply that the heuristic is favourable, in the same way as for the single bottleneck problem. The results show that the proposed heuristic yields better results than integer programming, especially in the five-end-item problems, where they are very close to the linear programming solutions. The reason will be explained in chapter 6.

As a result, a simple heuristic for the bottleneck multi-level lot-sizing problem has been developed, which provides quick and easy solutions for the problem and is sufficiently simple to be used even without a computer routine. More discussion will be given in chapter 6.

Table 4.6.1. One End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
50% Util.	Current Value	10869	15441	13070	13183
	Iterations	144	3332	-	-
	CPU Time	0.80	127.98	1.20	1.20
	Optimality*	1.000	1.420	1.202	1.212
75% Util.	Current Value	10869	16010	13977	14090
	Iterations	151	1684	-	-
	CPU Time	0.84	135.58	1.21	1.21
	Optimality*	1.000	1.472	1.285	1.296
95% Util.	Current Value	10898	16044	14814	14927
	Iterations	148	2393	-	-
	CPU Time	0.78	105.15	1.22	1.22
	Optimality*	1.000	1.472	1.359	1.369

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.1. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Heuristic Solution** Silver-Meal
50% Util.	Current Value	8695	12352	10456	10546
	Iterations	144	3332	-	-
	CPU Time	0.78	129.36	1.20	1.20
	Optimality*	1.000	1.420	1.202	1.212
75% Util.	Current Value	8695	12808	11182	11272
	Iterations	151	1684	-	-
	CPU Time	0.88	134.18	1.21	1.21
	Optimality*	1.000	1.472	1.285	1.296
95% Util.	Current Value	8718	12835	11851	11941
	Iterations	148	2393	-	-
	CPU Time	0.86	102.76	1.22	1.22
	Optimality*	1.000	1.472	1.359	1.369

* The proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% reduction in the Original Costs

Table 4.6.1. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
	Current Value	6521	9264	7842	7910
50%	Iterations	144	3332	-	-
Util.	CPU Time	0.76	127.18	1.21	1.21
	Optimality*	1.000	1.420	1.202	1.212
	Current Value	6521	9606	8386	8454
75%	Iterations	151	1684	-	-
Util.	CPU Time	0.76	130.68	1.21	1.21
	Optimality*	1.000	1.472	1.285	1.296
	Current Value	6538	9626	8888	8956
95%	Iterations	148	2393	-	-
Util.	CPU Time	0.90	101.02	1.22	1.22
	Optimality*	1.000	1.472	1.359	1.369

* The proportion is calculated as Current Value/LP Current Value

*** Calculated for a 40% reduction in the Original Costs

Table 4.6.2. One End Item Problem with Structure Two

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Silver-Meal
	Current Value	30036	36235	35785	34749
50%	Iteration	144	1713	-	-
Util.	CPU Time	0.80	60.36	3.10	3.10
	Optimality*	1.000	1.206	1.191	1.156
	Current Value	30036	38110	37293	36257
75%	Iterations	154	2420	-	-
Util.	CPU Time	0.88	78.92	3.11	3.11
	Optimality*	1.000	1.268	1.241	1.207
	Current Value	30331	38498	38700	37664
95%	Iterations	150	1828	-	-
Util.	CPU Time	0.84	78.92	3.11	3.11
	Optimality*	1.000	1.269	1.275	1.241

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.2. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution ** EOQ	Silver-Meal
	Current Value	24029	28988	28628	27799
50%	Iterations	144	1713	-	-
Util.	CPU Time	0.86	60.45	3.10	3.10
	Optimality*	1.000	1.206	1.191	1.156
	Current Value	24029	30488	29834	29005
75%	Iterations	154	2420	-	-
Util.	CPU Time	0.90	79.66	3.11	3.11
	Optimality*	1.000	1.268	1.241	1.207
	Current Value	24265	30798	30960	30131
95%	Iterations	150	1828	-	-
Util.	CPU Time	0.80	60.74	3.11	3.11
	Optimality*	1.000	1.269	1.275	1.241

* The proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% reduction in the Original Costs

Table 4.6.2. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
	Current Value	18021	21741	21471	20849
50%	Iterations	144	1713	-	-
Util.	CPU Time	0.84	60.36	3.10	3.10
	Optimality*	1.000	1.206	1.191	1.156
	Current Value	18021	22866	22375	21754
75%	Iterations	154	1796	-	-
Util.	CPU Time	0.98	79.41	3.11	3.11
	Optimality*	1.000	1.268	1.241	1.207
	Current Value	18198	23099	23220	22598
95%	Iterations	150	1828	-	-
Util.	CPU Time	0.90	60.20	3.11	3.11
	Optimality*	1.000	1.269	1.275	1.241

* The proportion is calculated as Current Value/LP Current Value
*** Calculated for a 40% reduction in the Original Costs

Table 4.6.3. One End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
50% Util.	Current Value	18787	26102	23716	22987
	Iterations	150	2338	-	-
	CPU Time	0.94	77.23	3.12	3.12
	Optimality*	1.000	1.389	1.262	1.223
75% Util.	Current Value	18863	26434	24596	23867
	Iterations	147	1832	-	-
	CPU Time	0.76	50.52	3.12	3.12
	Optimality*	1.000	1.401	1.303	1.265
95% Util.	Current Value	19375	27378	25502	24773
	Iterations	141	2258	-	-
	CPU Time	0.84	65.65	3.13	3.13
	Optimality*	1.000	1.413	1.316	1.278

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.3. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Heuristic Solution** Silver-Meal
50% Util.	Current Value	15030	20882	18972	18389
	Iterations	150	2338	-	-
	CPU Time	0.86	77.92	3.12	3.12
	Optimality*	1.000	1.389	1.262	1.223
75% Util.	Current Value	15090	21147	19677	19093
	Iterations	147	1832	-	-
	CPU Time	0.78	50.59	3.13	3.13
	Optimality*	1.000	1.401	1.303	1.265
95% Util.	Current Value	15500	21902	20401	19818
	Iterations	141	2558	-	-
	CPU Time	0.78	66.98	3.13	3.13
	Optimality*	1.000	1.413	1.316	1.278

* The proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% reduction in the Original Costs

Table 4.6.3. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
50% Util.	Current Value	11272	15661	14229	13792
	Iterations	150	2338	-	-
	CPU Time	0.88	77.45	3.12	3.12
	Optimality*	1.000	1.389	1.262	1.223
75% Util.	Current Value	11318	15860	14757	14320
	Iterations	147	1832	-	-
	CPU Time	0.86	50.85	3.13	3.13
	Optimality*	1.000	1.401	1.303	1.265
95% Util.	Current Value	11625	16426	15301	14863
	Iterations	141	2258	-	-
	CPU Time	0.78	65.95	3.13	3.13
	Optimality*	1.000	1.413	1.316	1.278

* The proportion is calculated as Current Value/LP Current Value

*** Calculated for a 40% reduction in the Original Costs

Table 4.6.4. Three End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Silver-Meal
50% Util.	Current Value	73623	92083	87156	84962
	Iterations	430	1503	-	-
	CPU Time	5.78	359.12	3.28	3.28
	Optimality*	1.000	1.250	1.183	1.154
75% Util.	Current Value	73623	93072	87649	85455
	Iterations	445	1807	-	-
	CPU Time	5.62	344.77	3.29	3.29
	Optimality*	1.000	1.264	1.190	1.160
95% Util.	Current Value	73902	94043	91859	89665
	Iterations	460	769	-	-
	CPU Time	6.06	340.84	3.30	3.30
	Optimality*	1.000	1.272	1.242	1.213

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.4. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Silver-Meal
50% Util.	Current Value	58898	73666	69725	67970
	Iterations	430	1503	-	-
	CPU Time	5.69	368.22	3.28	3.28
	Optimality*	1.000	1.250	1.183	1.154
75% Util.	Current Value	58898	74458	70119	68364
	Iterations	445	1807	-	-
	CPU Time	5.56	341.39	3.29	3.29
	Optimality*	1.000	1.264	1.190	1.160
95% Util.	Current Value	59121	75234	73487	71732
	Iterations	460	769	-	-
	CPU Time	6.06	351.54	3.30	3.30
	Optimality*	1.000	1.272	1.242	1.213

* The proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% reduction in the Original Costs

Table 4.6.4. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
50% Util.	Current Value	44174	55249	52294	50977
	Iterations	430	1503	-	-
	CPU Time	5.28	319.18	3.29	3.29
	Optimality*	1.000	1.250	1.183	1.154
75% Util.	Current Value	44174	55843	52589	51273
	Iterations	445	1807	-	-
	CPU Time	5.67	353.58	3.29	3.29
	Optimality*	1.000	1.264	1.190	1.160
95% Util.	Current Value	44341	56426	55115	53799
	Iterations	460	769	-	-
	CPU Time	6.20	332.94	3.30	3.30
	Optimality*	1.000	1.272	1.242	1.213

* The proportion is calculated as Current Value/LP Current Value

*** Calculated for a 40% reduction in the Original Costs

Table 4.6.5. Three End Item Problem with Structure Two

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Silver-Meal
50% Util.	Current Value	78353	97496	99336	96098
	Iterations	427	1039	-	-
	CPU Time	5.22	312.01	3.14	3.14
	Optimality*	1.000	1.244	1.267	1.226
75% Util.	Current Value	78367	98608	100086	96848
	Iterations	435	663	-	-
	CPU Time	5.78	325.9	3.15	3.15
	Optimality*	1.000	1.258	1.277	1.235
95% Util.	Current Value	79163	102507	102629	99391
	Iterations	438	3625	-	-
	CPU Time	5.96	423.32	3.16	3.16
	Optimality*	1.000	1.294	1.296	1.255

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.5. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Silver-Meal
50% Util.	Current Value	62683	77997	79469	76879
	Iterations	427	1037	-	-
	CPU Time	5.34	342.8	3.15	3.15
	Optimality*	1.000	1.244	1.267	1.226
75% Util.	Current Value	62694	78886	80069	77478
	Iterations	435	663	-	-
	CPU Time	5.84	338.8	3.15	3.15
	Optimality*	1.000	1.258	1.277	1.235
95% Util.	Current Value	63330	82006	82103	79513
	Iterations	435	3610	-	-
	CPU Time	5.80	412.42	3.16	3.16
	Optimality*	1.000	1.294	1.296	1.255

* The proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% reduction in the Original Costs

Table 4.6.5. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
	Current Value	47012	58498	59602	57695
50%	Iterations	427	1037	-	-
Util.	CPU Time	5.70	374.2	3.15	3.15
	Optimality*	1.000	1.244	1.267	1.226
	Current Value	47020	59165	60051	58109
75%	Iterations	435	663	-	-
Util.	CPU Time	5.86	377.98	3.16	3.16
	Optimality*	1.000	1.258	1.277	1.235
	Current Value	47498	61462	61577	59635
95%	Iterations	435	4188	-	-
Util.	CPU Time	6.02	443.72	3.16	3.16
	Optimality*	1.000	1.294	1.296	1.255

* The proportion is calculated as Current Value/LP Current Value

*** Calculated for a 40% reduction in the Original Costs

Table 4.6.6. Three End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Silver-Meal
	Current Value	58424	75478	69989	68776
50%	Iterations	393	835	-	-
Util.	CPU Time	5.14	359.00	3.16	3.16
	Optimality*	1.000	1.291	1.197	1.177
	Current Value	59061	74549	69465	68252
75%	Iterations	431	2238	-	-
Util.	CPU Time	5.74	375.3	3.18	3.18
	Optimality*	1.000	1.262	1.176	1.155
	Current Value	61973	75723	69958	68745
95%	Iterations	433	1713	-	-
Util.	CPU Time	5.78	458.51	3.18	3.18
	Optimality*	1.000	1.221	1.128	1.109

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.6. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Silver-Meal
50% Util.	Current Value	46739	60382	55991	55020
	Iterations	393	835	-	-
	CPU Time	4.98	279.5	3.17	3.17
	Optimality*	1.000	1.291	1.197	1.177
75% Util.	Current Value	47249	59639	55572	54601
	Iterations	431	2238	-	-
	CPU Time	5.76	370.48	3.18	3.18
	Optimality*	1.000	1.262	1.176	1.155
95% Util.	Current Value	49579	60579	55966	54996
	Iterations	433	1705	-	-
	CPU Time	5.64	454.35	3.18	3.18
	Optimality*	1.000	1.221	1.128	1.109

* The proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% reduction in the Original Costs

Table 4.6.6. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
50% Util.	Current Value	35054	45287	41993	41265
	Iterations	393	835	-	-
	CPU Time	5.00	265.0	3.17	3.17
	Optimality*	1.000	1.291	1.197	1.177
75% Util.	Current Value	35436	44729	41679	40951
	Iterations	431	2238	-	-
	CPU Time	5.58	384.72	3.18	3.18
	Optimality*	1.000	1.262	1.176	1.155
95% Util.	Current Value	37184	45434	41975	41247
	Iterations	433	1716	-	-
	CPU Time	5.92	448.23	3.18	3.18
	Optimality*	1.000	1.221	1.128	1.109

* The proportion is calculated as Current Value/LP Current Value
*** Calculated for a 40% reduction in the Original Costs

Table 4.6.7. Five End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
50% Util.	Current Value	128948	158390	136935	133173
	Iterations	699	2797	-	-
	CPU Time	14.50	675.85	1.07	1.07
	Optimality*	1.000	1.228	1.061	1.032
75% Util.	Current Value	128948	159198	136838	133076
	Iterations	721	2864	-	-
	CPU Time	14.19	592.60	1.08	1.08
	Optimality*	1.000	1.234	1.061	1.032
95% Util.	Current Value	129108	162936	137141	133379
	Iterations	774	3825	-	-
	CPU Time	16.28	684.9	1.09	1.09
	Optimality*	1.000	1.262	1.062	1.033

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.7. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Heuristic Solution** Silver-Meal
50% Util.	Current Value	103158	126712	109548	106538
	Iterations	699	3156	-	-
	CPU Time	14.02	665.00	1.08	1.08
	Optimality*	1.000	1.228	1.061	1.032
75% Util.	Current Value	103158	127358	109470	106460
	Iterations	721	3486	-	-
	CPU Time	14.16	573.24	1.08	1.08
	Optimality*	1.000	1.234	1.061	1.032
95% Util.	Current Value	103286	130348	109713	106703
	Iterations	774	3825	-	-
	CPU Time	15.92	681.7	1.09	1.09
	Optimality*	1.000	1.262	1.062	1.033

* The proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% reduction in the Original Costs

Table 4.6.7. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
50% Util.	Current Value	77369	95034	82161	79904
	Iterations	699	3156	-	-
	CPU Time	14.18	612.14	1.08	1.08
	Optimality*	1.000	1.228	1.061	1.032
75% Util.	Current Value	77369	95518	82102	79845
	Iterations	721	3486	-	-
	CPU Time	14.10	566.78	1.09	1.09
	Optimality*	1.000	1.234	1.061	1.032
95% Util.	Current Value	77465	97761	82284	80027
	Iterations	774	3825	-	-
	CPU Time	15.72	680.54	1.09	1.09
	Optimality*	1.000	1.262	1.062	1.033

* The proportion is calculated as Current Value/LP Current Value

*** Calculated for a 40% reduction in the Original Costs

Table 4.6.8. Five End Item Problem with Structure Two

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Silver-Meal
50% Util.	Current Value	107807	131277	108211	107889
	Iterations	697	2610	-	-
	CPU Time	13.69	596.20	2.35	2.35
	Optimality*	1.000	1.217	1.003	1.000~
75% Util.	Current Value	107889	134606	108853	107889
	Iterations	685	1216	-	-
	CPU Time	13.97	751.6	2.36	2.36
	Optimality*	1.000	1.247	1.008	1.000~
95% Util.	Current Value	109141	134673	111845	109141
	Iterations	694	3526	-	-
	CPU Time	15.12	665.65	2.36	2.36
	Optimality*	1.000	1.233	1.024	1.000~

* The proportion is calculated as Current Value/LP Current Value

~ Taking into account the rounding error

Table 4.6.8. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Silver-Meal
	Current Value	86246	105022	86569	86246
50%	Iterations	697	2610	-	-
Util.	CPU Time	14.06	612.9	2.35	2.35
	Optimality*	1.000	1.217	1.003	1.000~
	Current Value	86311	107684	87082	86311
75%	Iterations	685	1216	-	-
Util.	CPU Time	14.30	732.5	2.36	2.36
	Optimality*	1.000	1.247	1.008	1.000~
	Current Value	87313	107657	89476	87313
95%	Iterations	694	4445	-	-
Util.	CPU Time	15.33	639.77	2.37	2.37
	Optimality*	1.000	1.233	1.024	1.000~

* The proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% reduction in the Original Costs
~ Taking into account the rounding error

Table 4.6.8. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
	Current Value	64684	78766	64926	64684
50%	Iterations	697	2610	-	-
Util.	CPU Time	14.03	638.30	2.35	2.35
	Optimality*	1.000	1.217	1.003	1.000~
	Current Value	64733	80763	65311	64733
75%	Iterations	685	1216	-	-
Util.	CPU Time	13.52	663.72	2.36	2.36
	Optimality*	1.000	1.247	1.008	1.000~
	Current Value	65484	80742	67107	65484
95%	Iterations	699	3526	-	-
Util.	CPU Time	20.50	620.52	2.37	2.37
	Optimality*	1.000	1.233	1.024	1.000~

* The proportion is calculated as Current Value/LP Current Value
*** Calculated for a 40% reduction in the Original Costs
~ Taking into account the rounding error

Table 4.6.9. Five End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
	Current Value	95601	120031	100383	95774
50%	Iterations	676	3294	-	-
Util.	CPU Time	12.66	478.9	1.23	1.23
	Optimality*	1.000	1.255	1.050	1.001
	Current Value	96722	122075	100403	96722
75%	Iterations	496	1287	-	-
Util.	CPU Time	14.02	621.28	1.24	1.24
	Optimality*	1.000	1.262	1.038	1.000~
	Current Value	99591	125047	101426	99591
95%	Iterations	729	3239	-	-
Util.	CPU Time	15.56	672.26	1.25	1.25
	Optimality*	1.000	1.255	1.018	1.000~

* The proportion is calculated as Current Value/LP Current Value
~ Taking into account the rounding error

Table 4.6.9. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Heuristic Solution** Silver-Meal
	Current Value	76481	96025	80306	76619
50%	Iterations	736	3295	-	-
Util.	CPU Time	14.04	523.52	1.24	1.24
	Optimality*	1.000	1.255	1.050	1.001
	Current Value	77377	97660	80322	77377
75%	Iterations	713	1298	-	-
Util.	CPU Time	14.02	620.02	1.24	1.24
	Optimality*	1.000	1.262	1.038	1.000~
	Current Value	79673	100037	81140	79673
95%	Iterations	751	3223	-	-
Util.	CPU Time	16.34	648.19	1.25	1.25
	Optimality*	1.000	1.255	1.018	1.000~

* The proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% reduction in the Original Costs
~ Taking into account the rounding error

Table 4.6.9. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
50% Util.	Current Value	57361	72019	60229	57361
	Iterations	713	3230	-	-
	CPU Time	13.94	508.58	1.24	1.24
	Optimality*	1.000	1.255	1.050	1.001
75% Util.	Current Value	58033	73245	60242	57476
	Iterations	702	1292	-	-
	CPU Time	13.52	615.05	1.24	1.24
	Optimality*	1.000	1.262	1.038	1.000~
95% Util.	Current Value	59754	75028	60855	58090
	Iterations	750	3270	-	-
	CPU Time	15.50	643.85	1.25	1.25
	Optimality*	1.000	1.255	1.018	1.000~

* The proportion is calculated as Current Value/LP Current Value

*** Calculated for a 40% reduction in the Original Costs

~ Taking into account the rounding error

Table 4.6.10. One End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
50% Util.	Current Value	10869	15441	13070	13183
	Iterations	144	3332	-	-
	CPU Time	0.80	127.98	1.20	1.20
	Optimality*	1.000	1.420	1.202	1.212
75% Util.	Current Value	10869	16010	13977	14090
	Iterations	151	1684	-	-
	CPU Time	0.84	135.58	1.21	1.21
	Optimality*	1.000	1.472	1.285	1.296
95% Util.	Current Value	10898	16044	14814	14927
	Iterations	148	2393	-	-
	CPU Time	0.78	105.15	1.22	1.22
	Optimality*	1.000	1.472	1.359	1.369

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.10. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Heuristic Solution** Silver-Meal
50% Util.	Current Value	9483	13820	11316	11785
	Iterations	145	1107	-	-
	CPU Time	0.76	117.32	1.20	1.20
	Optimality*	1.000	1.457	1.193	1.242
75% Util.	Current Value	9483	14268	12222	12691
	Iterations	150	1101	-	-
	CPU Time	0.86	25.48	1.21	1.21
	Optimality*	1.000	1.504	1.288	1.338
95% Util.	Current Value	9506	15234	13071	13540
	Iterations	148	2815	-	-
	CPU Time	0.80	111.66	1.22	1.22
	Optimality*	1.000	1.602	1.374	1.424

* The Proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% Reduction in the Original Holding costs

Table 4.6.10. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
50% Util.	Current Value	10081	13981	12688	12202
	Iterations	143	2868	-	-
	CPU Time	0.80	93.69	1.20	1.20
	Optimality*	1.000	1.386	1.258	1.210
75% Util.	Current Value	10081	14268	13416	12929
	Iterations	149	1101	-	-
	CPU Time	0.84	25.48	1.21	1.21
	Optimality*	1.000	1.415	1.330	1.282
95% Util.	Current Value	10110	14687	14072	13586
	Iterations	147	2971	-	-
	CPU Time	0.86	132.08	1.22	1.22
	Optimality*	1.000	1.452	1.391	1.343

* The Proportion is calculated as Current Value/LP Current Value

*** Calculated for a 20% Reduction in the Original Setup costs

Table 4.6.11. Three End Item Problem with Structure Two

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Silver-Meal
50% Util.	Current Value	78353	97496	99336	96098
	Iterations	427	1039	-	-
	CPU Time	5.22	312.01	3.14	3.14
	Optimality*	1.000	1.244	1.267	1.226
75% Util.	Current Value	78367	98608	100086	96848
	Iterations	435	663	-	-
	CPU Time	5.78	325.9	3.15	3.15
	Optimality*	1.000	1.258	1.277	1.235
95% Util.	Current Value	79163	102507	102629	99391
	Iterations	438	3625	-	-
	CPU Time	5.96	423.32	3.16	3.16
	Optimality*	1.000	1.294	1.296	1.255

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.11. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Silver-Meal
	Current Value	65458	82120	82729	80274
50%	Iterations	405	597	-	-
Util.	CPU Time	5.74	322.47	3.15	3.15
	Optimality*	1.000	1.254	1.263	1.226
	Current Value	65469	84162	83709	81254
75%	Iterations	436	681	-	-
Util.	CPU Time	6.24	360.38	3.15	3.15
	Optimality*	1.000	1.285	1.278	1.241
	Current Value	66105	85396	86543	84088
95%	Iterations	438	2557	-	-
Util.	CPU Time	5.96	371.80	3.16	3.16
	Optimality*	1.000	1.291	1.309	1.272

* The Proportion is calculated as Current Value/LP Current Value

** Calculated for a 20% Reduction in the Original Holding costs

Table 4.6.11. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
	Current Value	75578	92643	95359	91966
50%	Iterations	415	1462	-	-
Util.	CPU Time	5.46	328.91	3.15	3.15
	Optimality*	1.000	1.225	1.261	1.216
	Current Value	75592	93509	95729	92336
75%	Iterations	420	1132	-	-
Util.	CPU Time	5.32	366.46	3.15	3.15
	Optimality*	1.000	1.237	1.266	1.221
	Current Value	76388	94994	97472	94079
95%	Iterations	429	1072	-	-
Util.	CPU Time	5.98	356.54	3.16	3.16
	Optimality*	1.000	1.243	1.276	1.231

* The Proportion is calculated as Current Value/LP Current Value

*** Calculated for a 20% Reduction in the Original Setup costs

Table 4.6.12. Five End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
50% Util.	Current Value	95601	120031	100383	95774
	Iterations	676	3294	-	-
	CPU Time	12.66	478.9	1.23	1.23
	Optimality*	1.000	1.255	1.050	1.001
75% Util.	Current Value	96722	122075	100403	96722
	Iterations	729	1287	-	-
	CPU Time	14.02	621.28	1.24	1.24
	Optimality*	1.000	1.262	1.038	1.000~
95% Util.	Current Value	99591	125047	101426	99591
	Iterations	729	3239	-	-
	CPU Time	15.56	672.26	1.25	1.25
	Optimality*	1.000	1.255	1.018	1.000~

* The Proportion is calculated as Current Value/LP Current Value
~ Taking into account the rounding error

Table 4.6.12. Continued

		LP Sol.**	IP Sol.**	Heuristic Solution** EOQ	Heuristic Solution** Silver-Meal
50% Util.	Current Value	81567	104444	83373	81567
	Iterations	696	1213	-	-
	CPU Time	13.61	639.54	1.21	1.21
	Optimality*	1.000	1.280	1.022	1.000~
75% Util.	Current Value	82463	107717	83696	82463
	Iterations	698	1164	-	-
	CPU Time	13.80	401.8	1.24	1.24
	Optimality*	1.000	1.306	1.014	1.000~
95% Util.	Current Value	84758	108854	84974	84758
	Iterations	733	1808	-	-
	CPU Time	14.68	747.40	1.25	1.25
	Optimality*	1.000	1.284	1.002	1.000~

* The Proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% Reduction in the Original Holding costs
~ Taking into account the rounding error

Table 4.6.12. Continued

		LP Sol.***	IP Sol.***	Heuristic Solution*** EOQ	Silver-Meal
	Current Value	90516	110948	97949	92544
50%	Iterations	659	1103	-	-
Util.	CPU Time	13.48	613.94	1.21	1.21
	Optimality*	1.000	1.225	1.082	1.022
	Current Value	91636	114129	97663	92258
75%	Iterations	704	1530	-	-
Util.	CPU Time	13.80	521.00	1.24	1.24
	Optimality*	1.000	1.245	1.065	1.006
	Current Value	94506	115501	98225	94506
95%	Iterations	760	2073	-	-
Util.	CPU Time	15.10	587.96	1.25	1.25
	Optimality*	1.000	1.222	1.039	1.000~

* The Proportion is calculated as Current Value/LP Current Value

*** Calculated for a 20% Reduction in the Original Setup costs

~ Taking into account the rounding error

Table 4.6.13. One End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
60% Util.	Current Value	10869	15852	13173	13826
	Iterations	150	1459	-	-
	CPU Time	0.82	54.70	1.02	1.02
	Optimality*	1.000	1.458	1.261	1.272
80% Util.	Current Value	10869	16396	14277	14390
	Iterations	149	1370	-	-
	CPU Time	7.36	51.22	1.02	1.02
	Optimality*	1.000	1.508	1.313	1.323

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.14. One End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
60% Util.	Current Value	18787	26469	24313	23584
	Iterations	154	843	-	-
	CPU Time	1.02	159.4	0.90	0.90
	Optimality*	1.000	1.408	1.294	1.255
80% Util.	Current Value	19035	26210	24891	24164
	Iterations	149	2935	-	-
	CPU Time	1.06	189.00	0.90	0.90
	Optimality*	1.000	1.376	1.307	1.269

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.15. Three End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
60% Util.	Current Value	73623	92215	88048	85854
	Iterations	471	1281	-	-
	CPU Time	7.36	168.60	1.49	1.49
	Optimality*	1.000	1.252	1.195	1.166
80% Util.	Current Value	73625	92777	88841	86647
	Iterations	477	2137	-	-
	CPU Time	8.10	218.33	1.49	1.49
	Optimality*	1.000	1.260	1.206	1.176

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.16. Three End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
60% Util.	Current Value	58564	75675	69747	68534
	Iterations	431	689	-	-
	CPU Time	6.46	453.54	0.44	0.44
	Optimality*	1.000	1.292	1.190	1.170
80% Util.	Current Value	59773	76037	69974	68761
	Iterations	423	762	-	-
	CPU Time	6.96	674.78	0.45	0.45
	Optimality*	1.000	1.272	1.170	1.150

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.17. Five End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
60% Util.	Current Value	128948	158884	136895	133134
	Iterations	717	1035	-	-
	CPU Time	18.16	305.32	2.30	2.30
	Optimality*	1.000	1.232	1.061	1.032
80% Util.	Current Value	128948	159982	136771	133009
	Iterations	745	2431	-	-
	CPU Time	20.44	410.38	2.30	2.30
	Optimality*	1.000	1.240	1.060	1.031

* The Proportion is calculated as Current Value/LP Current Value

Table 4.6.18. Five End Item Problem with Structure Three

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
60% Util.	Current Value	95645	121630	100167	95645
	Iterations	800	1284	-	-
	CPU Time	16.80	723.88	2.31	2.31
	Optimality*	1.000	1.271	1.047	1.000~
80% Util.	Current Value	97715	124207	100106	97715
	Iterations	702	1341	-	-
	CPU Time	16.06	887.08	3.31	3.31
	Optimality*	1.000	1.271	1.024	1.000~

* The Proportion is calculated as Current Value/LP Current Value

~ Taking into account the rounding error

Table 4.6.19. One End Item Problem with Structure One

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
	Current Value	10869	15446	13076	13201
50%+	Iterations	145	3012	-	-
25%.	CPU Time	0.84	97.77	1.02	1.02
Util.	Optimality*	1.000	1.421	1.202	1.214
	Current Value	10869	17256	14339	14464
75%+	Iterations	152	3675	-	-
50%	CPU Time	0.90	103.58	1.02	1.02
Util.	Optimality*	1.000	1.587	1.319	1.330
	Current Value	10898	18069	15783	15908
95%+	Iterations	159	1530	-	-
75%	CPU Time	1.02	113.02	1.03	1.03
Util.	Optimality*	1.000	1.658	1.448	1.459

* The proportion is calculated as Current Value/LP Current Value

Table 4.6.20. Three End Item Problem with Structure Two

		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
	Current Value	78353	97496	106574	104578
50%+	Iterations	426	1036	-	-
25%.	CPU Time	5.80	286.26	1.49	1.49
Util.	Optimality*	1.000	1.244	1.360	1.334
	Current Value	78367	99467	101822	99826
75%+	Iterations	479	2864	-	-
50%	CPU Time	6.54	309.88	1.49	1.49
Util.	Optimality*	1.000	1.269	1.299	1.273
	Current Value	79163	103072	103801	101805
95%+	Iterations	452	1105	-	-
75%	CPU Time	6.54	348.11	1.49	1.49
Util.	Optimality*	1.000	1.302	1.311	1.286

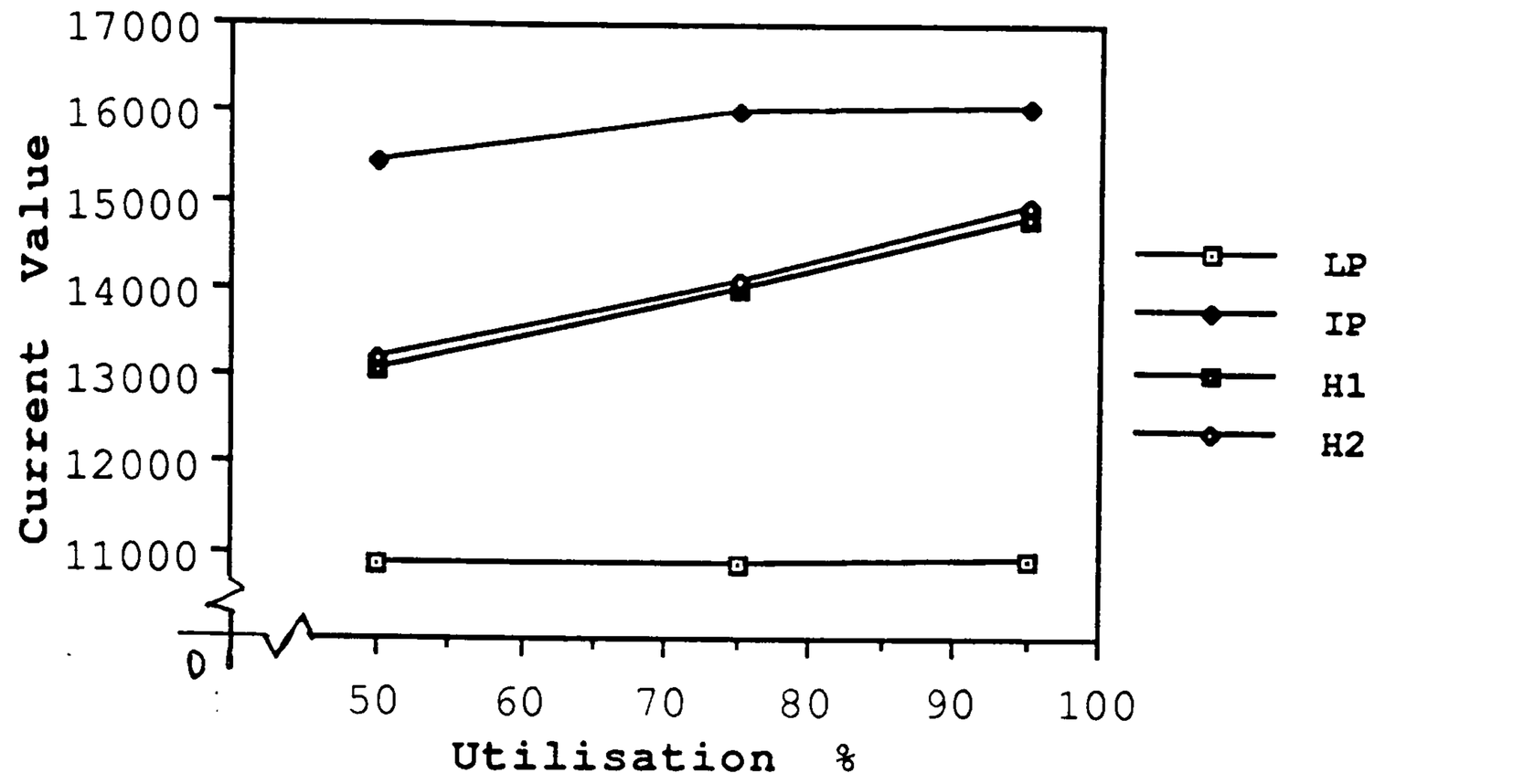
* The Proportion is calculated as Current Value/LP Current Value

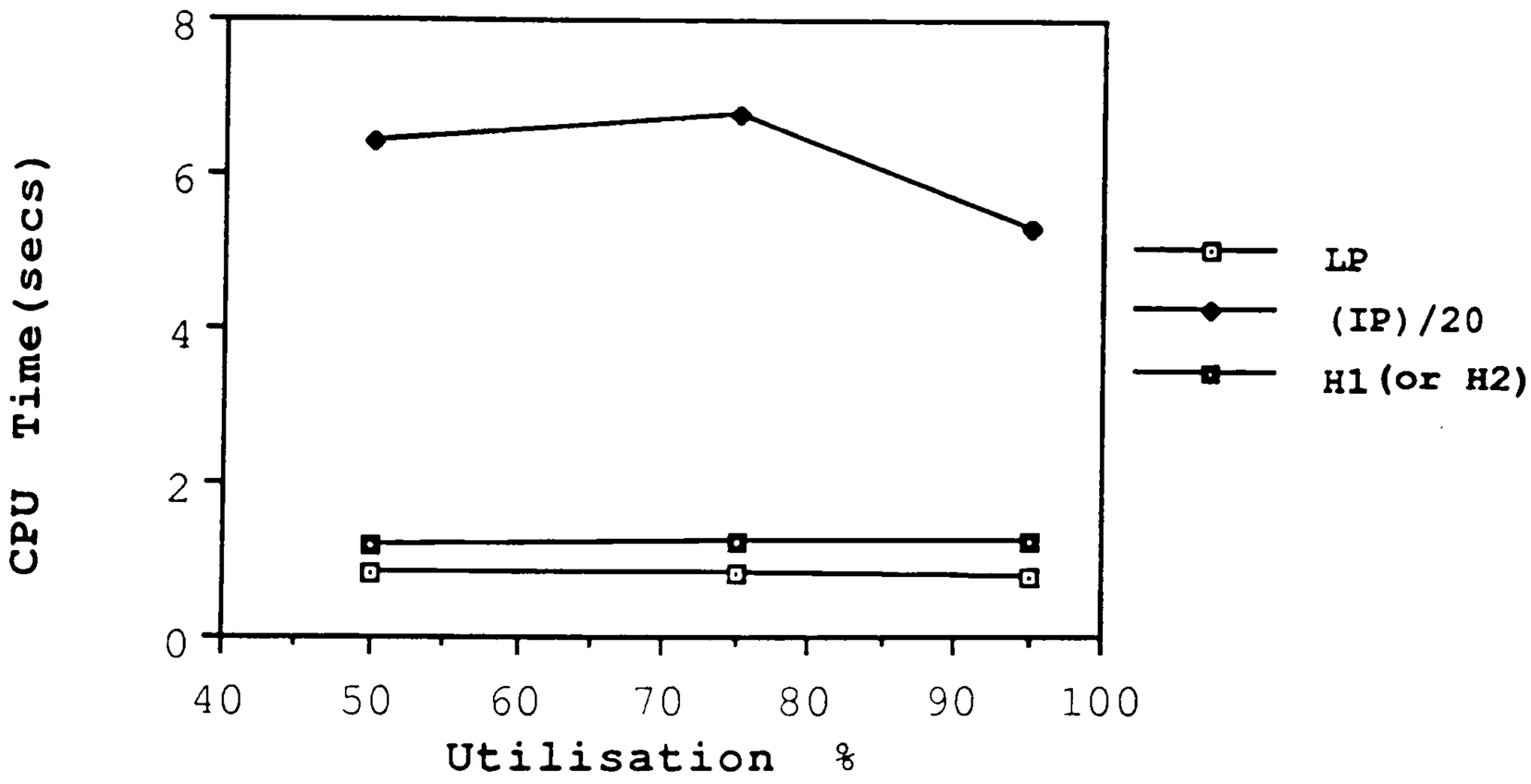
Table 4.6.21. Five End Item Problem with Structure Three

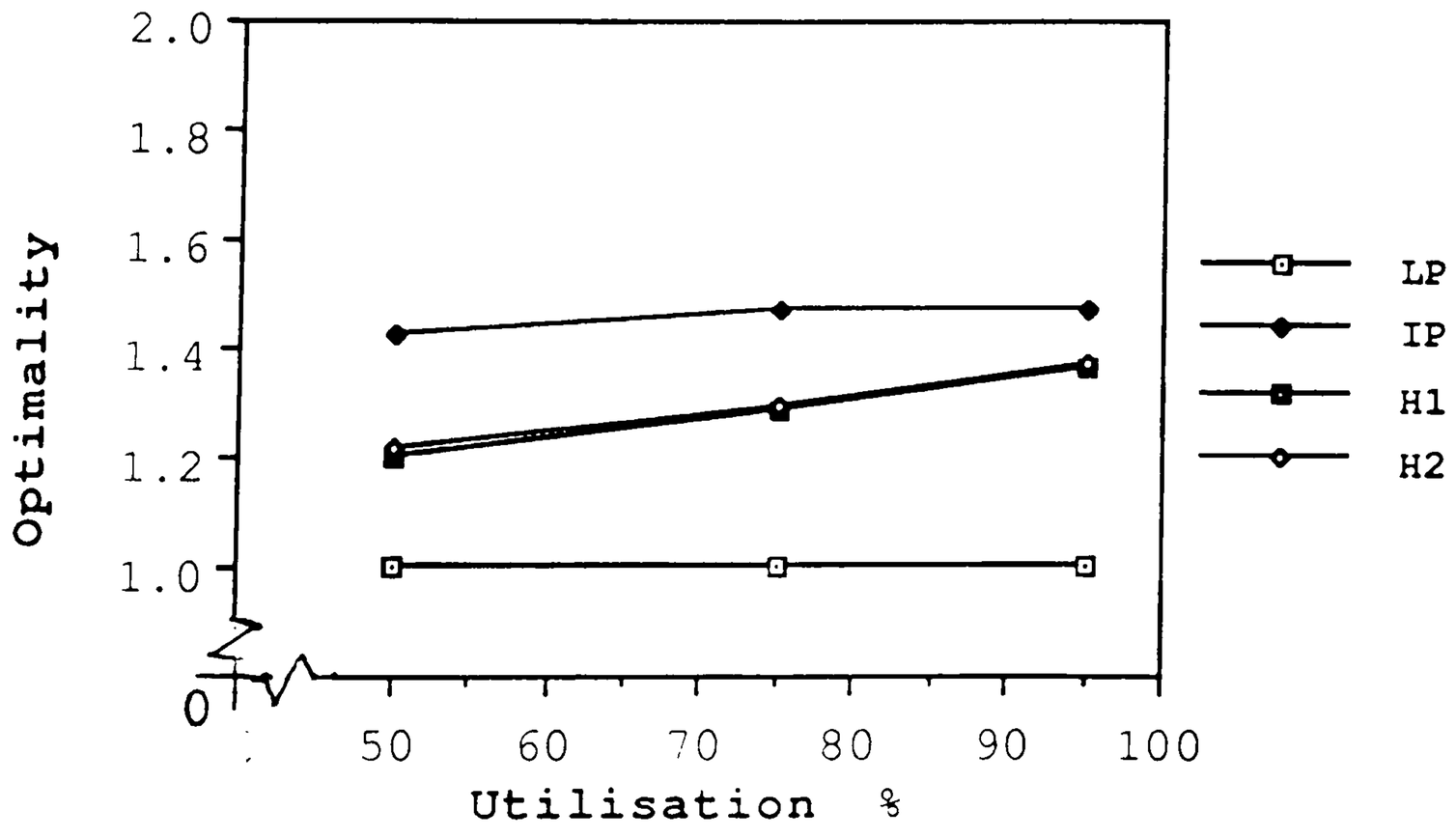
		LP Sol.	IP Sol.	Heuristic Solution EOQ	Heuristic Solution Silver-Meal
	Current Value	128948	158390	141841	138676
50%+	Iterations	726	2821	-	-
25%.	CPU Time	16.52	548.03	2.04	2.04
Util.	Optimality*	1.000	1.228	1.099	1.075
	Current Value	128948	160849	138281	135116
75%+	Iterations	775	1678	-	-
50%	CPU Time	17.82	566.47	2.04	2.04
Util.	Optimality*	1.000	1.247	1.072	1.047
	Current Value	129108	169130	138913	135748
95%	Iterations	783	3101	-	-
75%	CPU Time	18.62	679.64	2.04	2.04
Util.	Optimality	1.000	1.309	1.075	1.051

* The Proportion is calculated as Current Value/LP Current Value

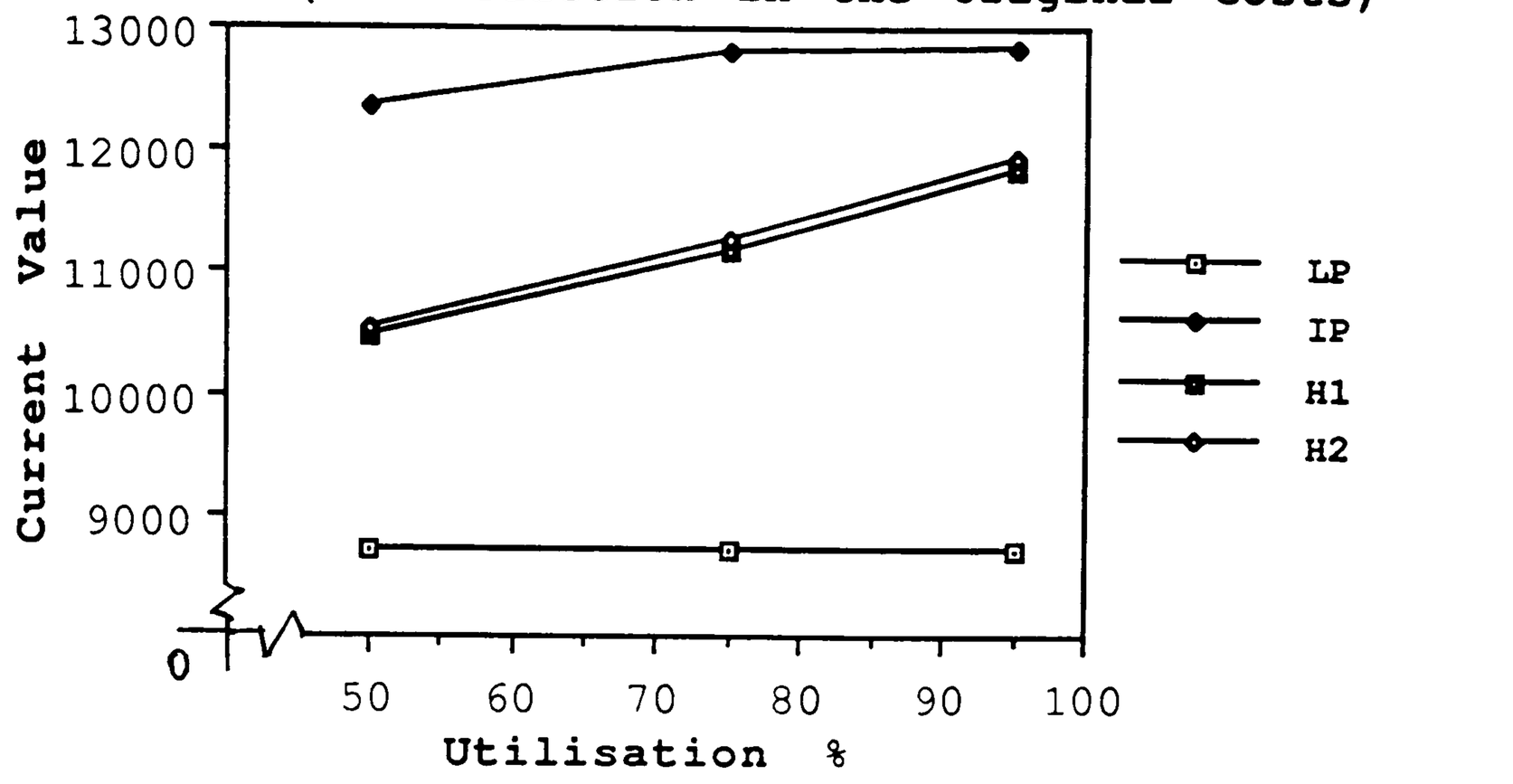
Graph 4.6.1. One End Item Problem with Structure 1

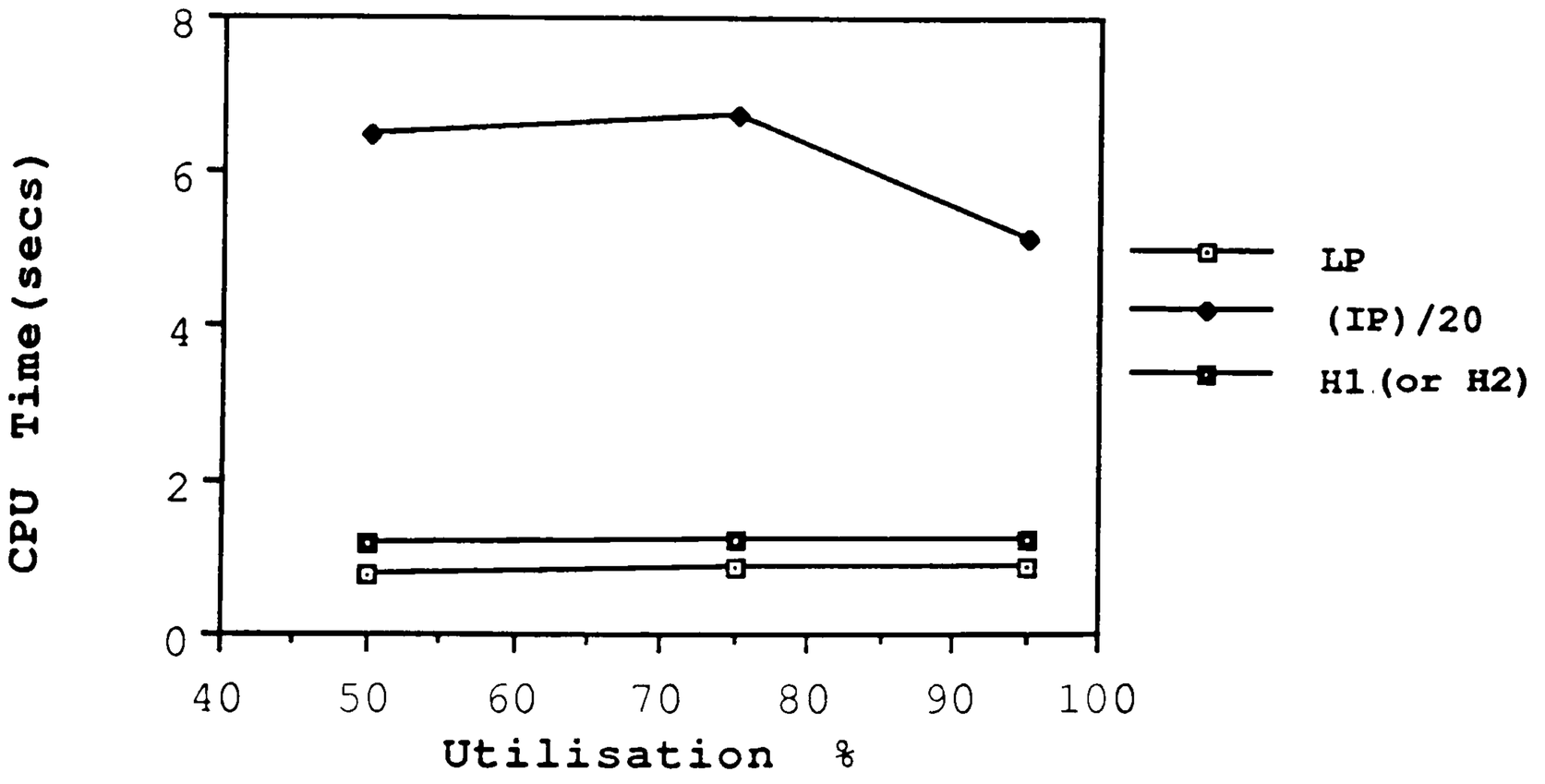


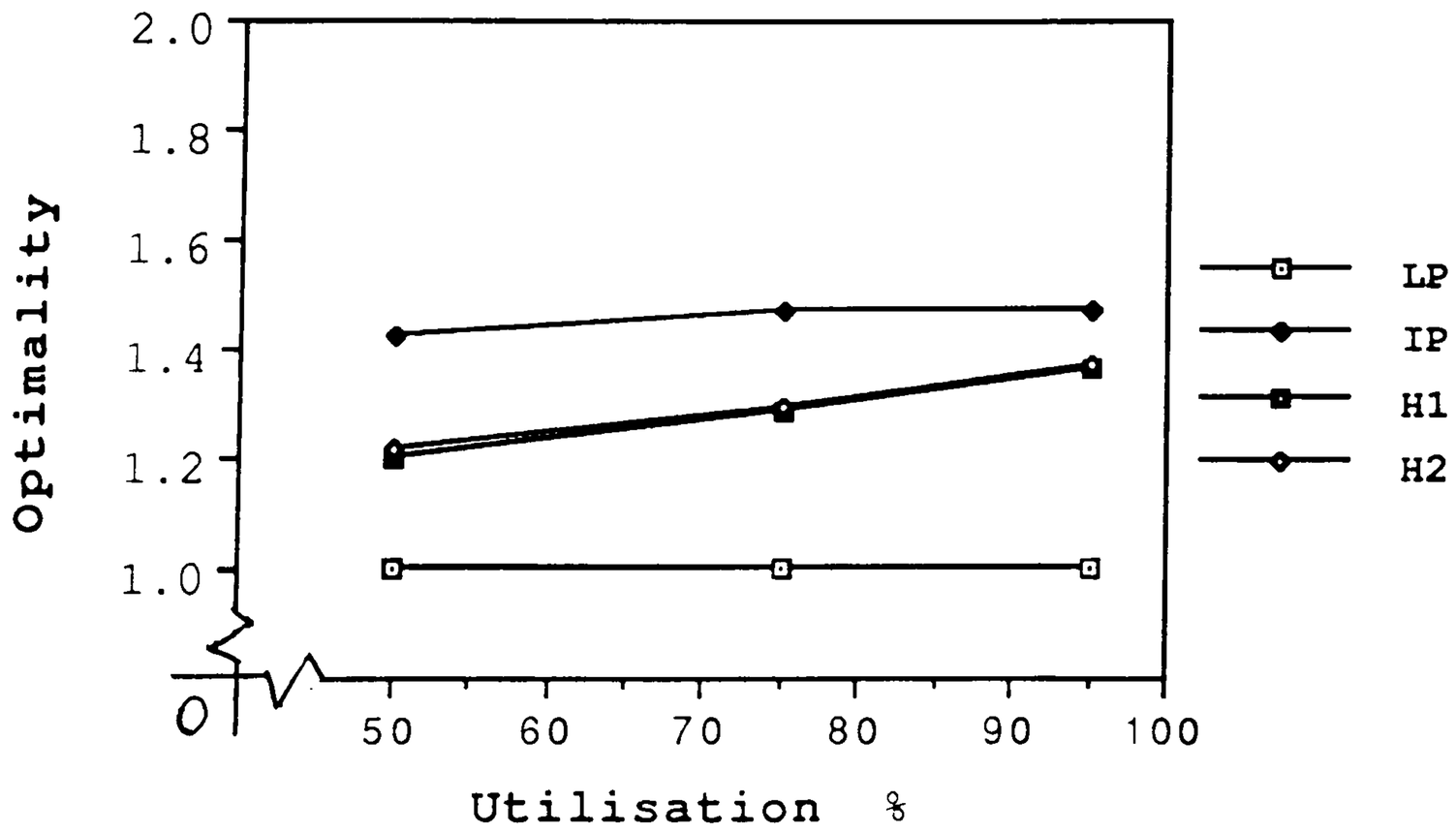




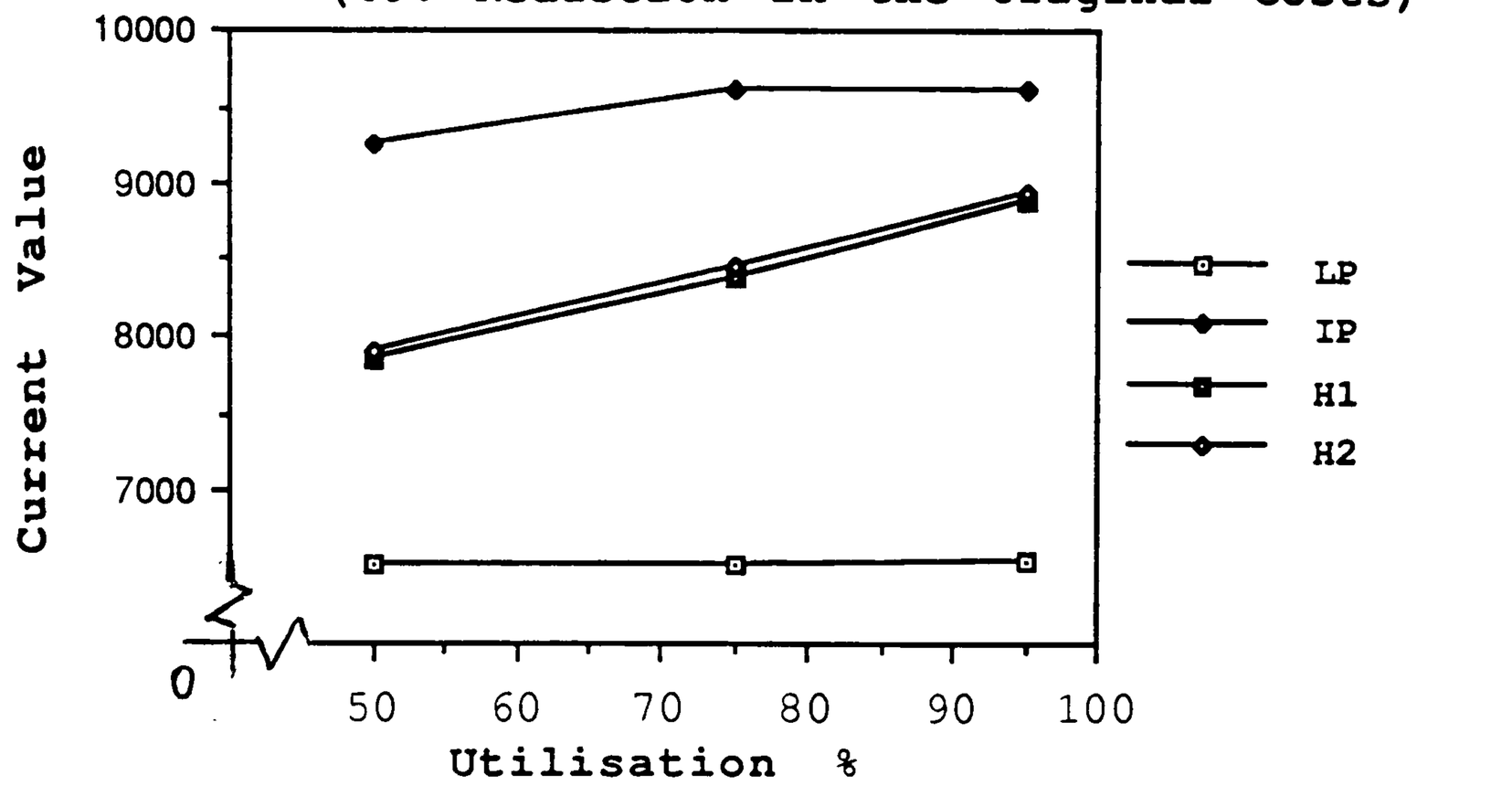
Graph 4.6.2. One End Item Problem with Structure 1
(20% Reduction in the Original Costs)

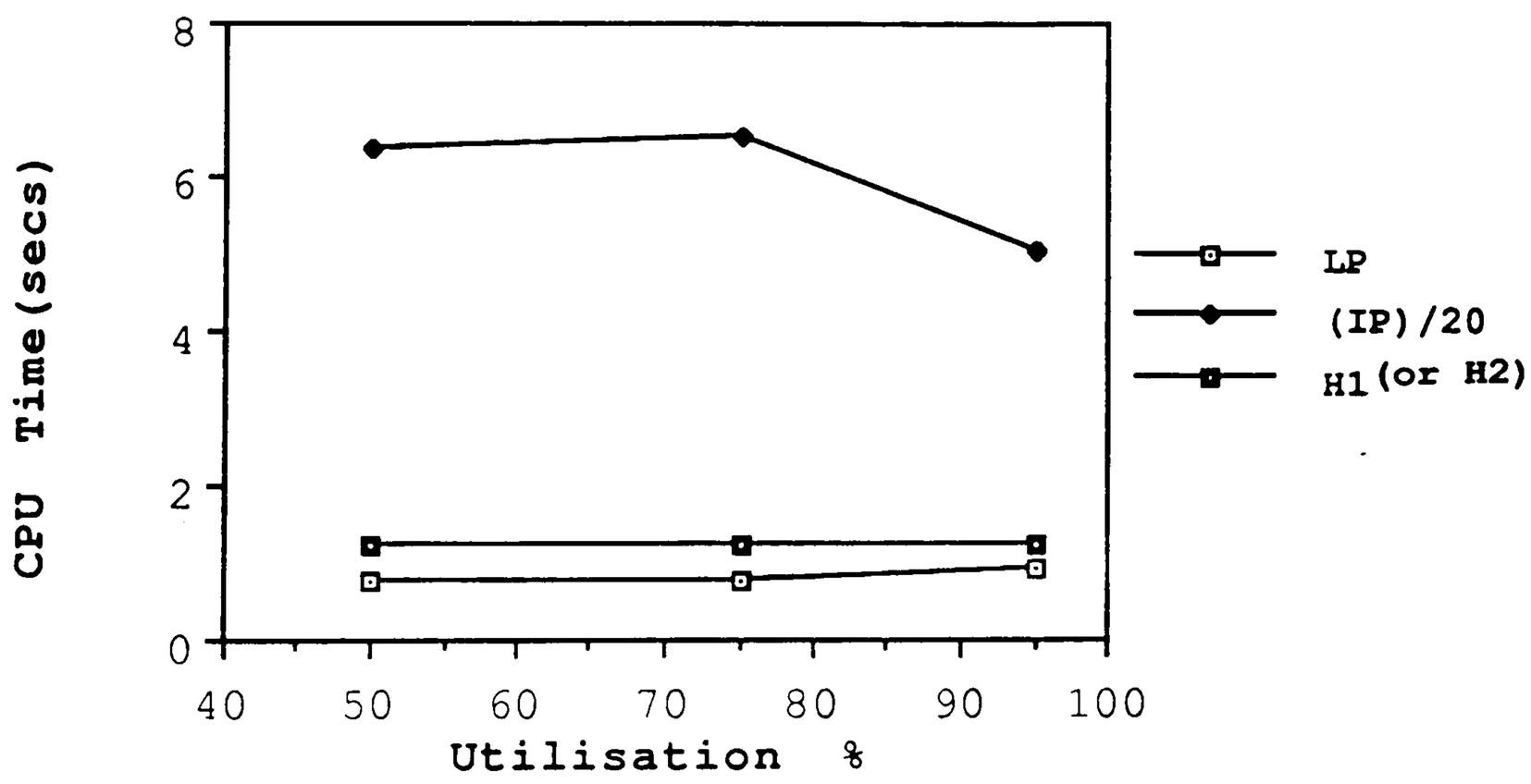


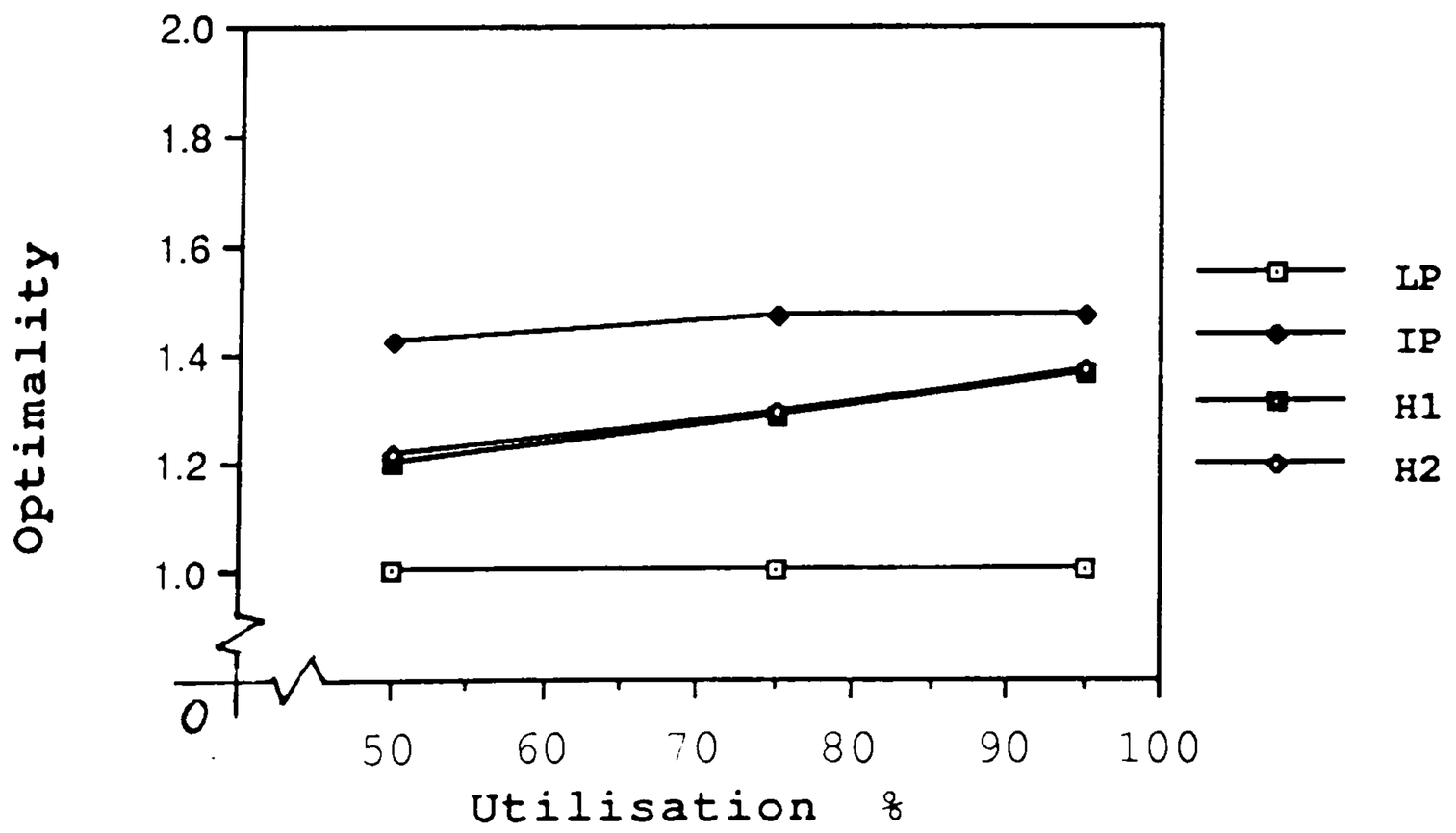




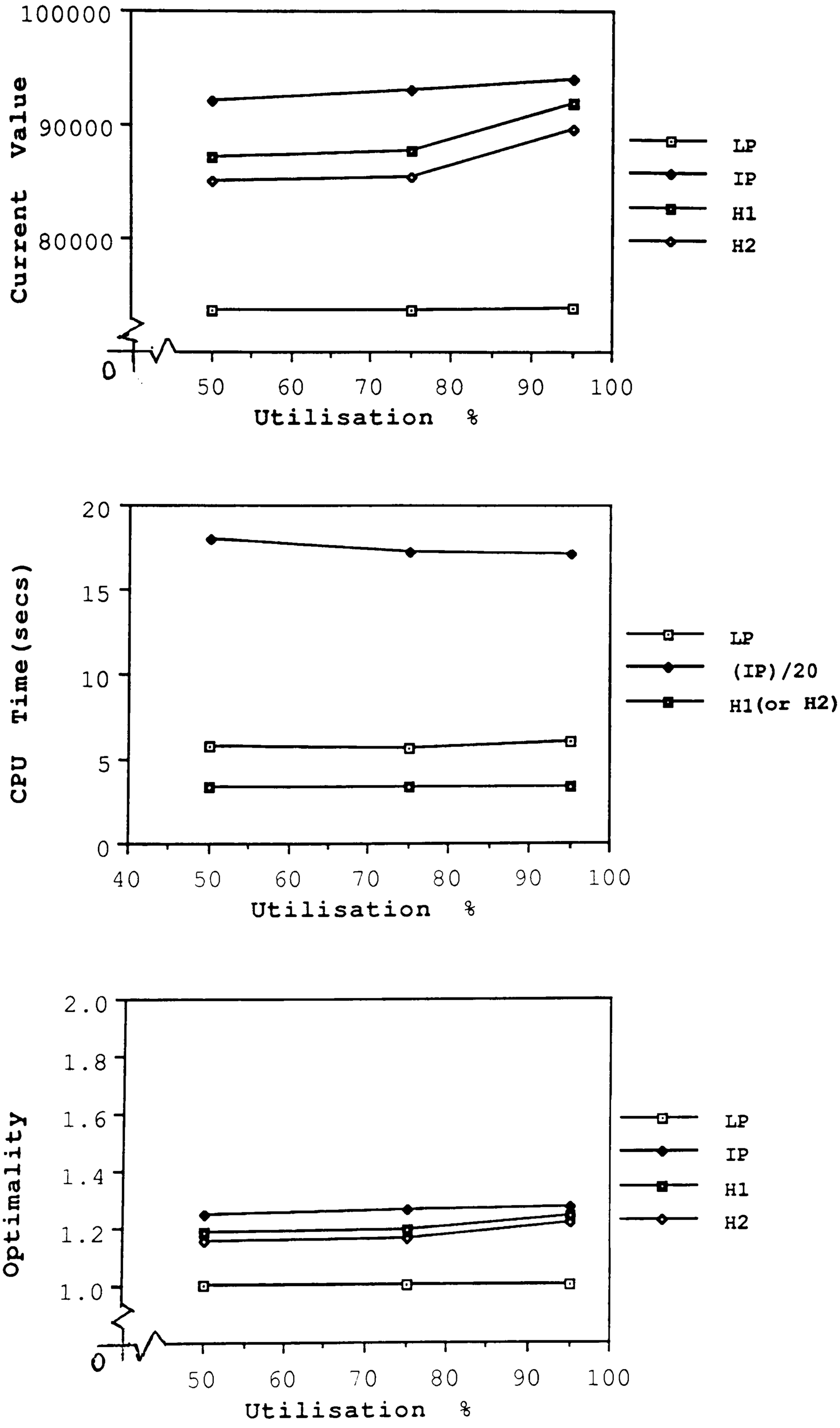
Graph 4.6.3. One End Item Problem with Structure 1
(40% Reduction in the Original Costs)



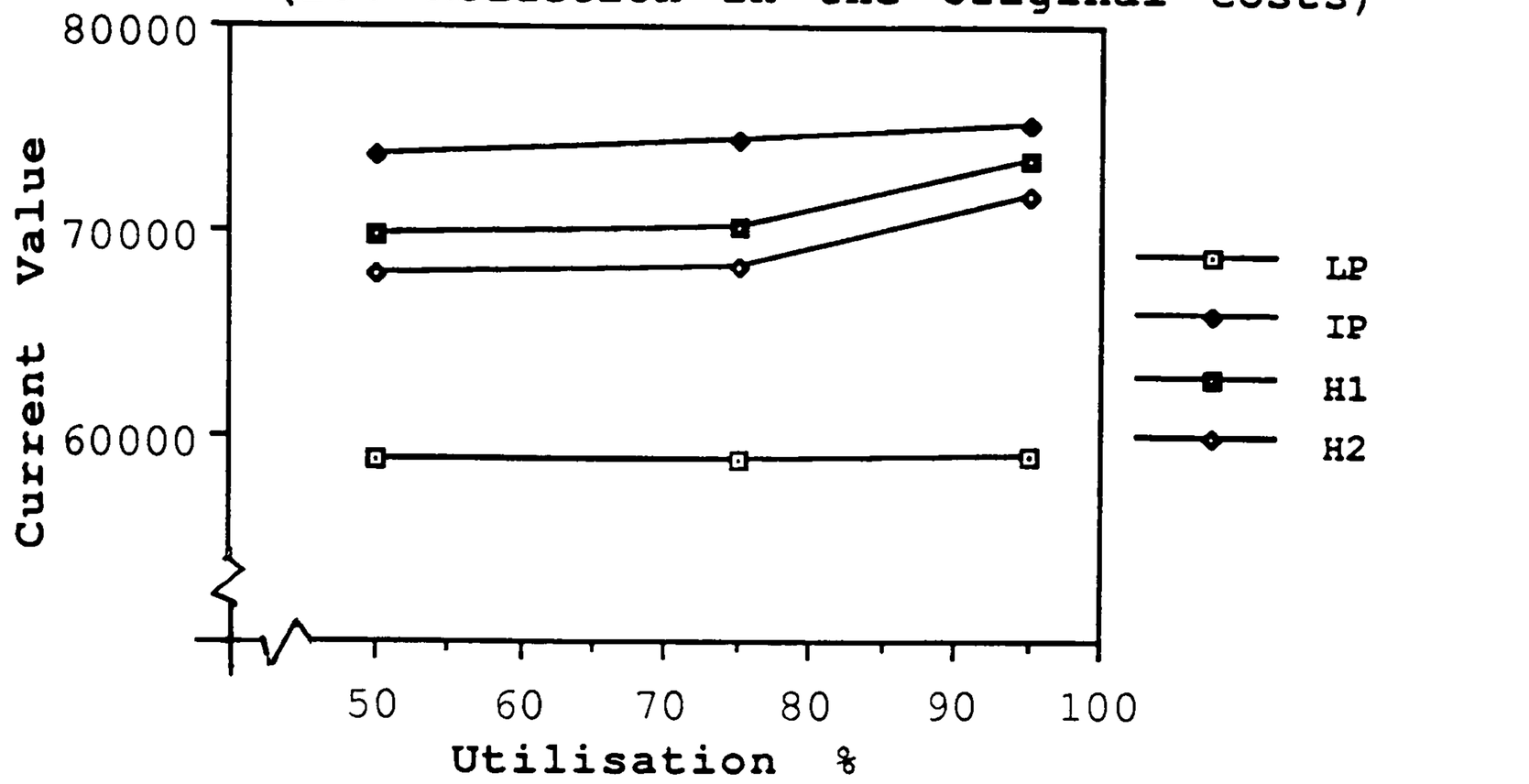


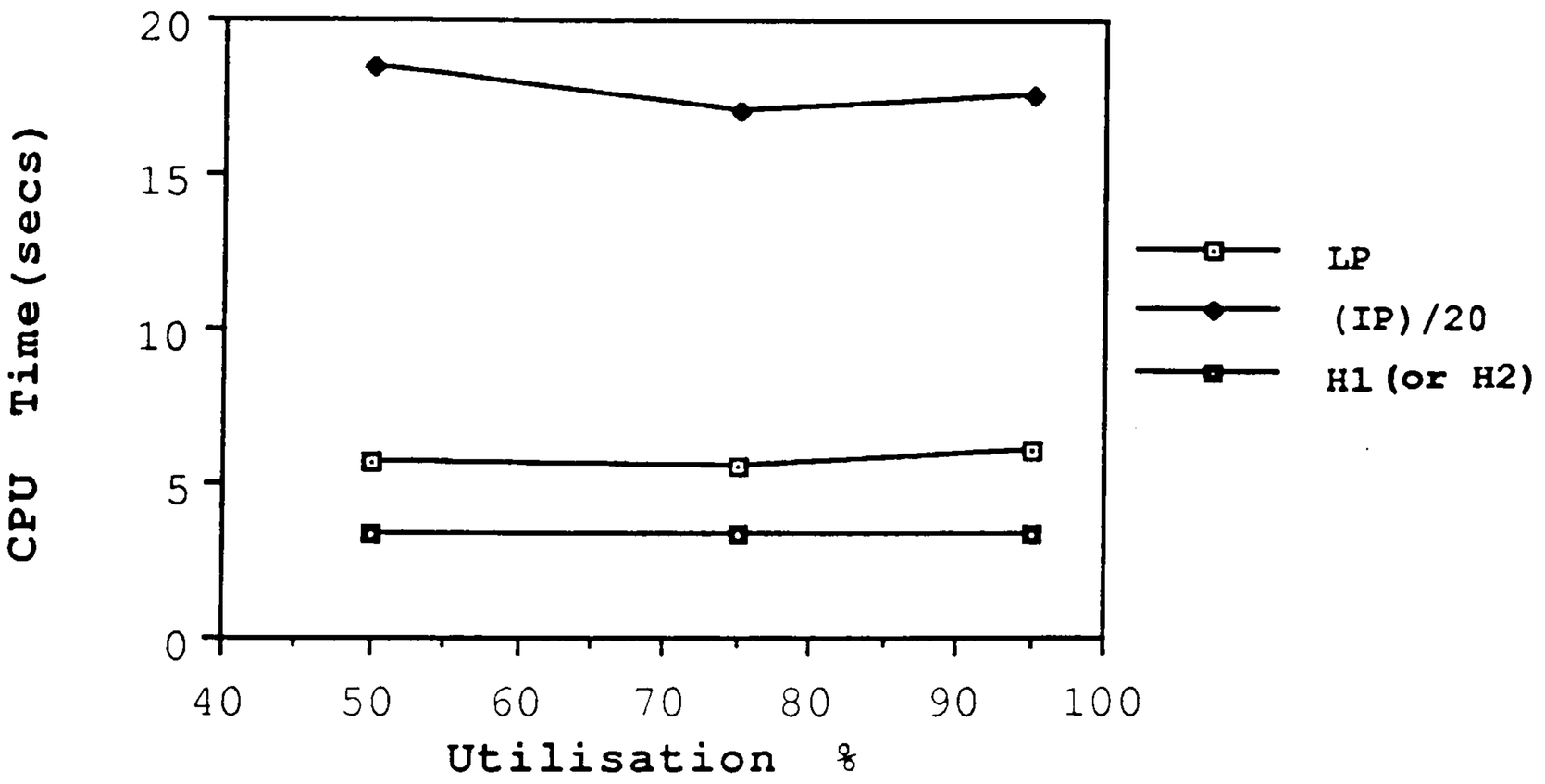


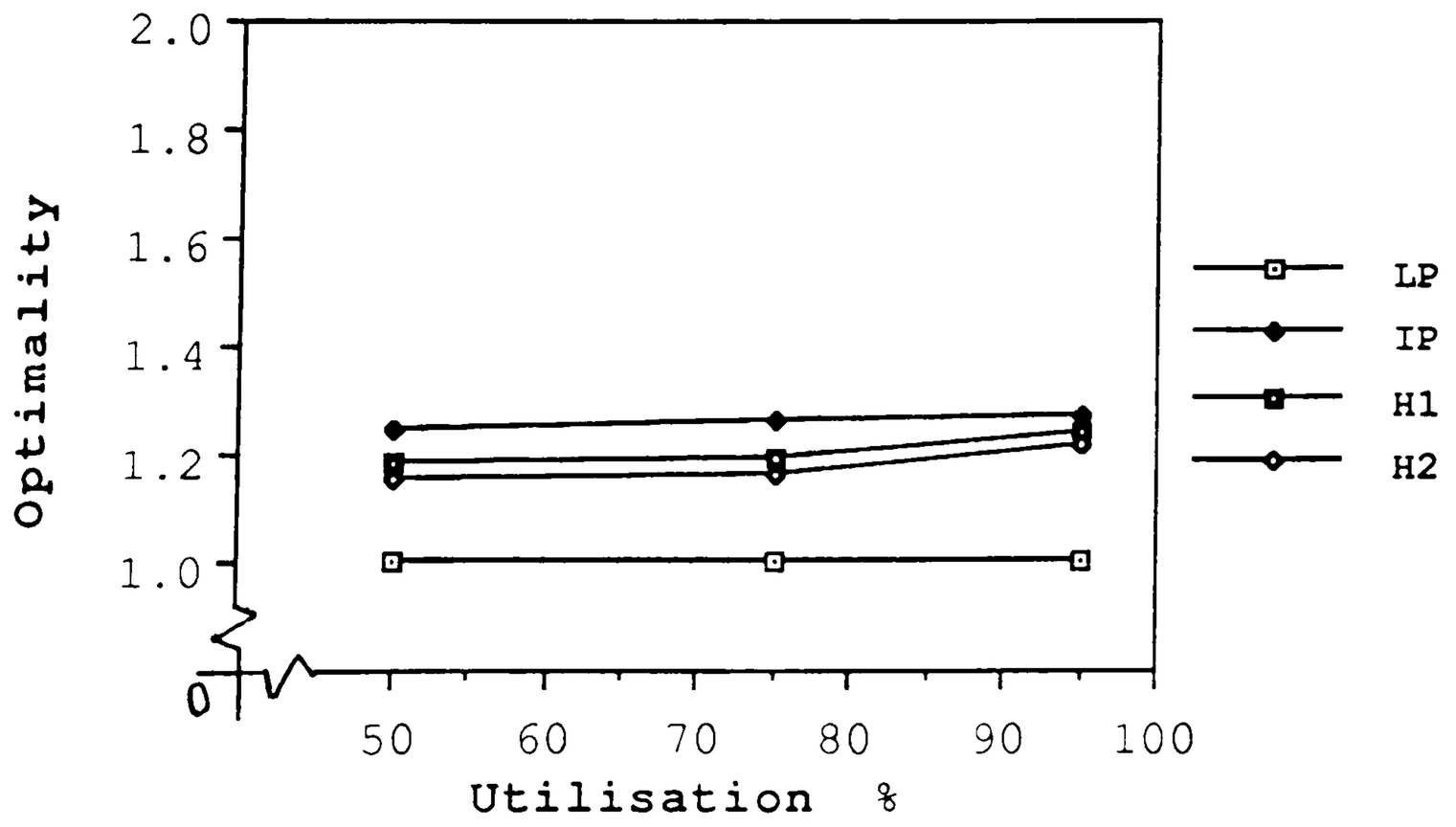
Graph 4.6.4.Three End Item Problem with Structure 1



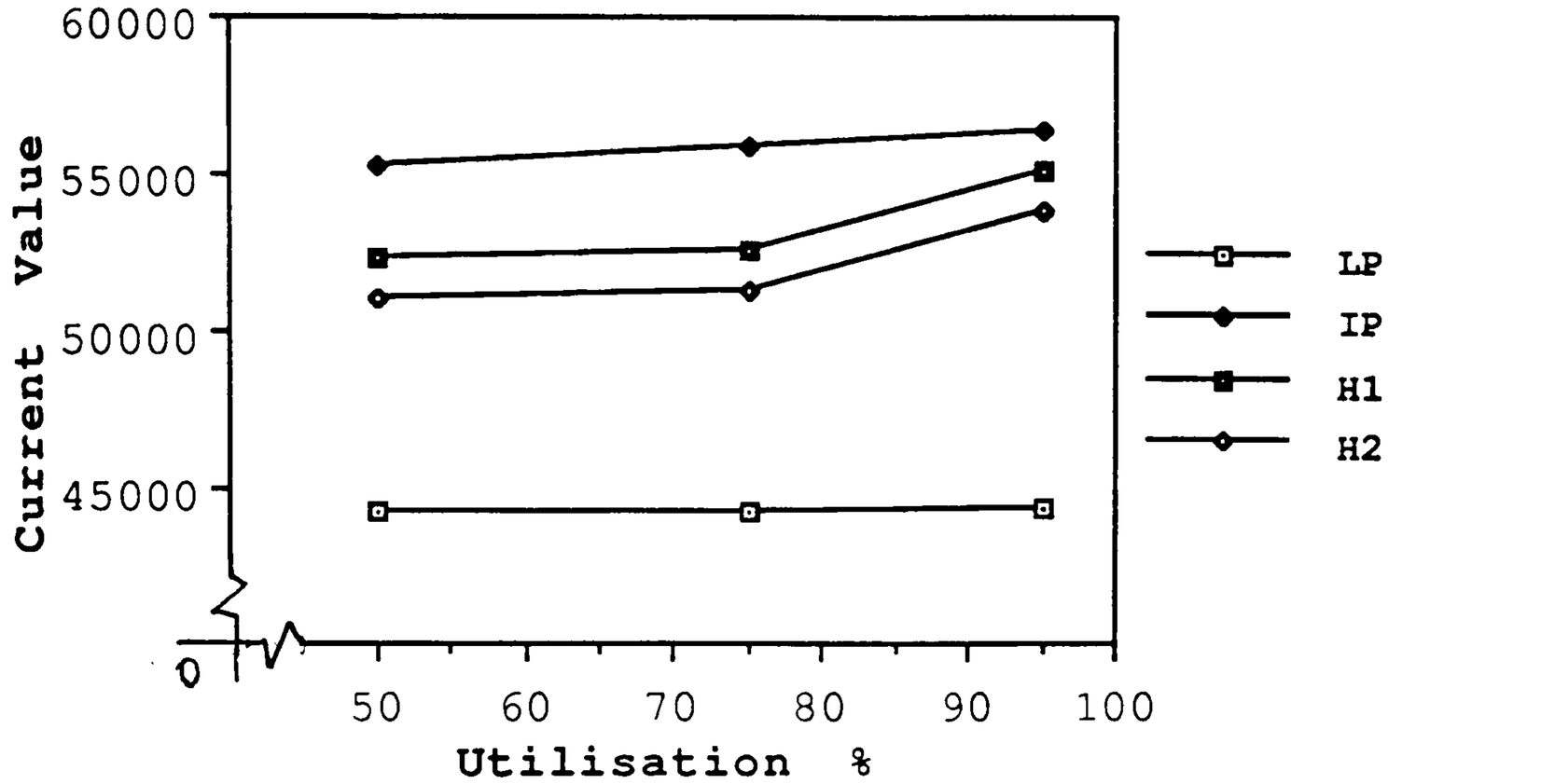
Graph 4.6.5.Three End Item Problem with Structure 1
(20% Reduction in the original costs)

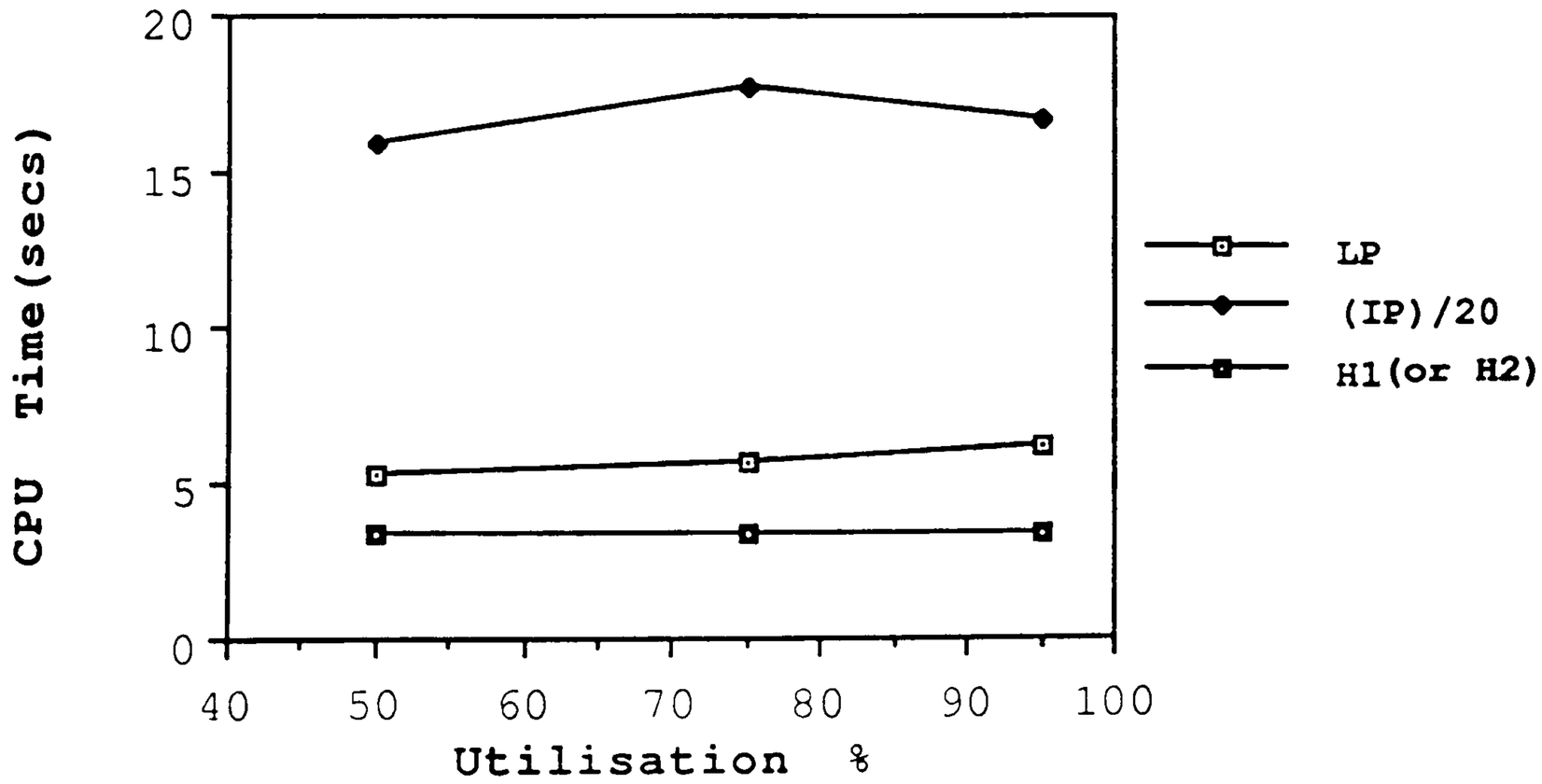


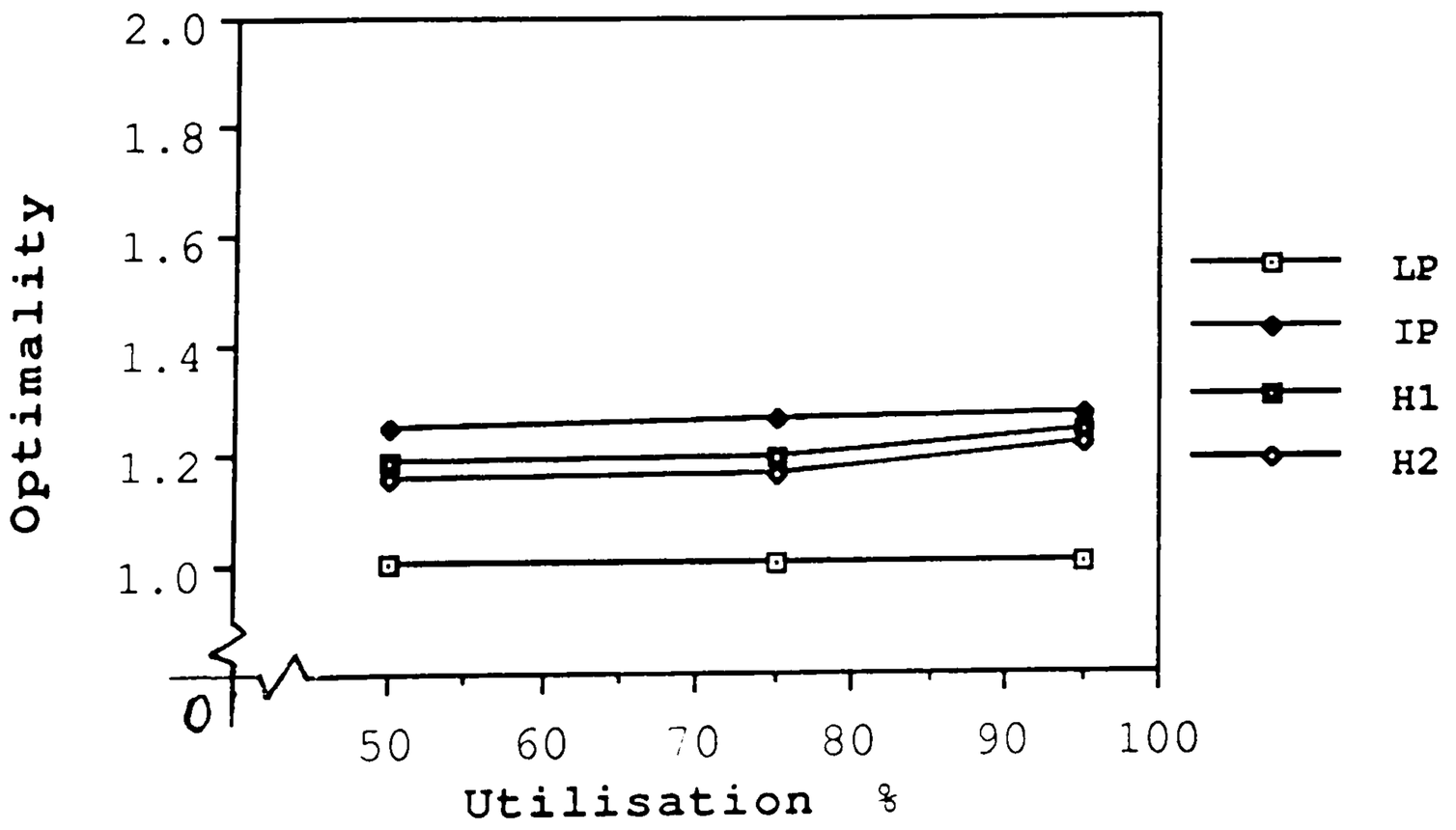




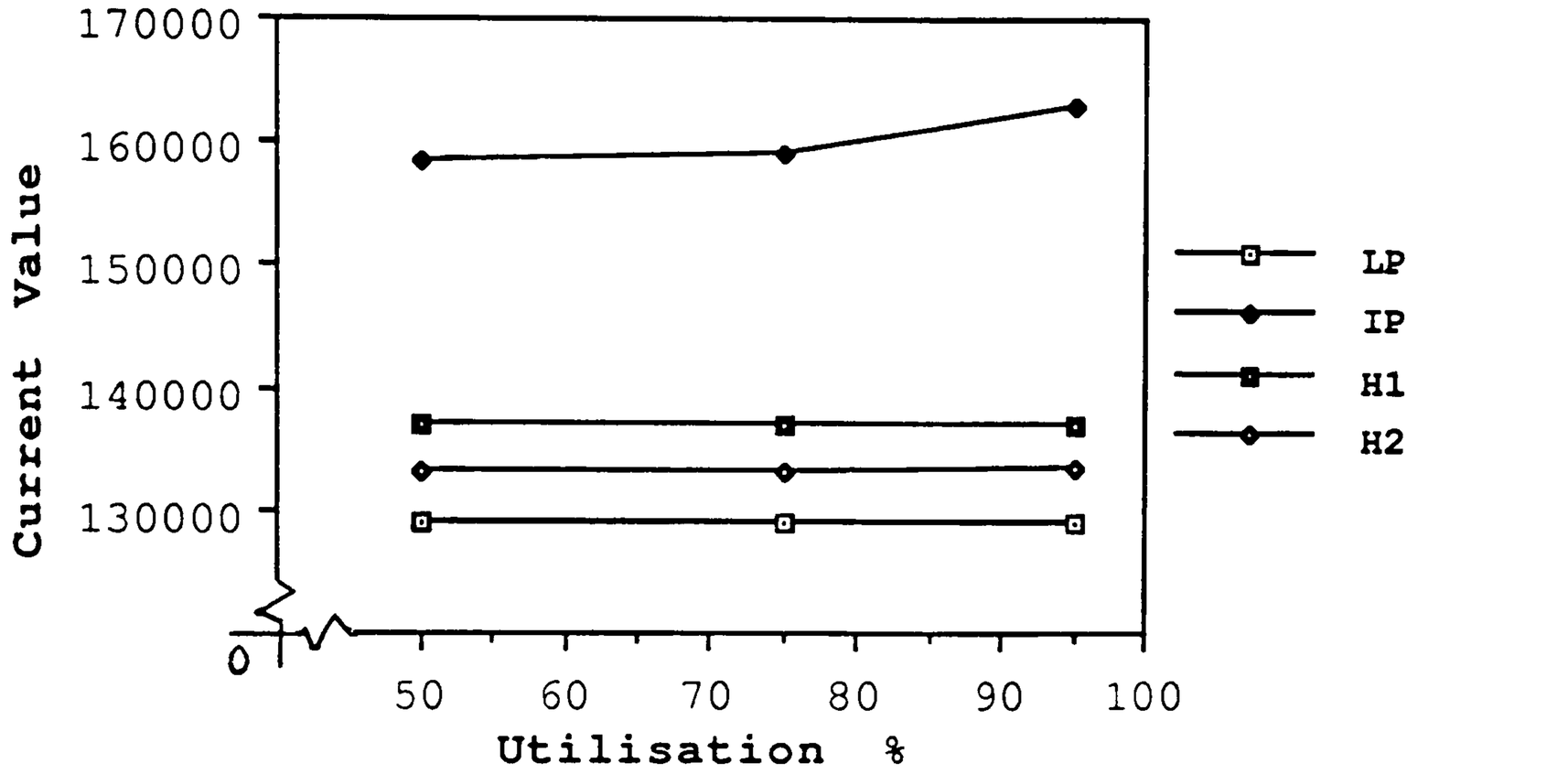
Graph 4.6.6.Three End Item Problem with Structure 1
(40% Reduction in the Original Costs)

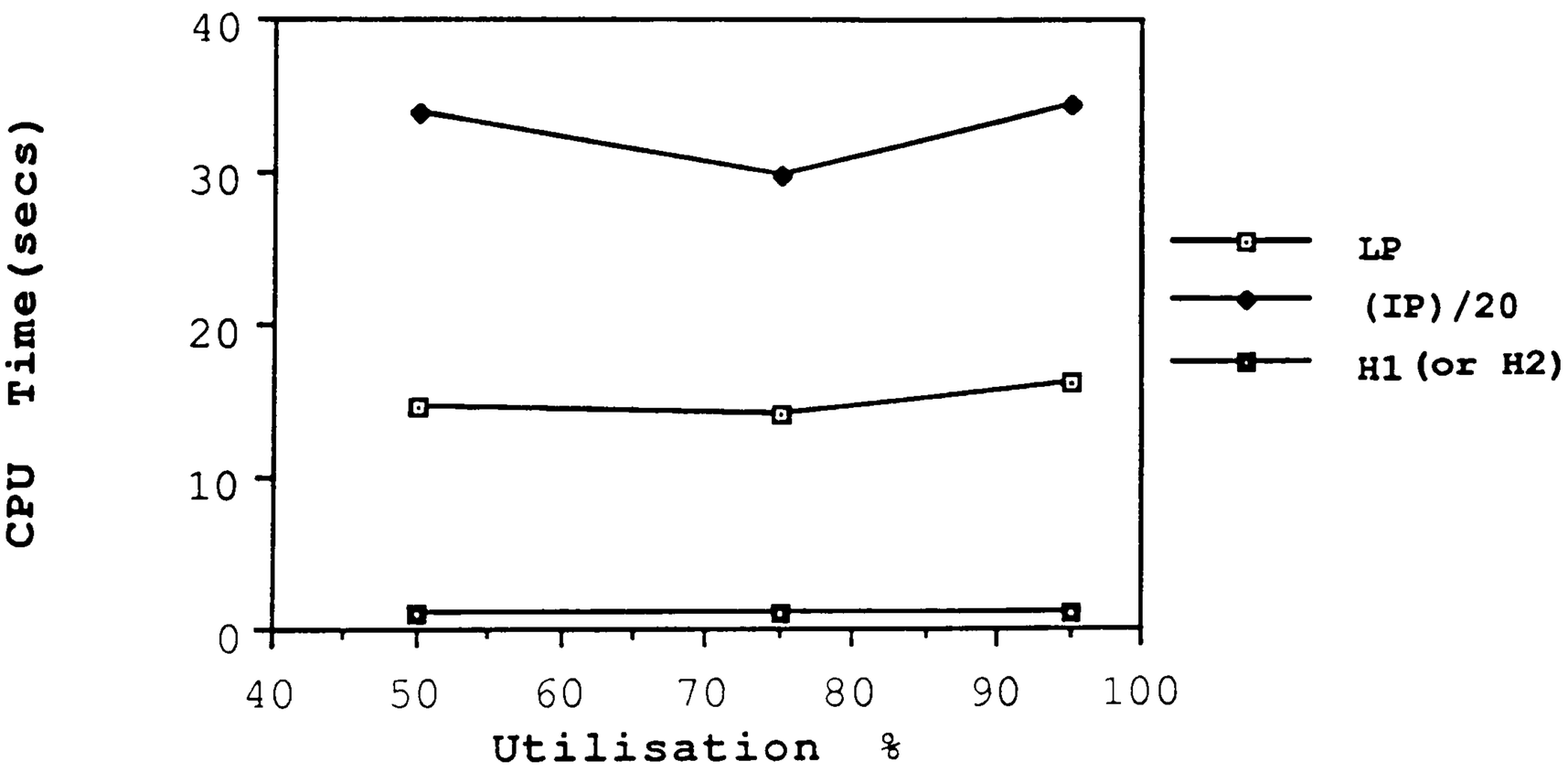


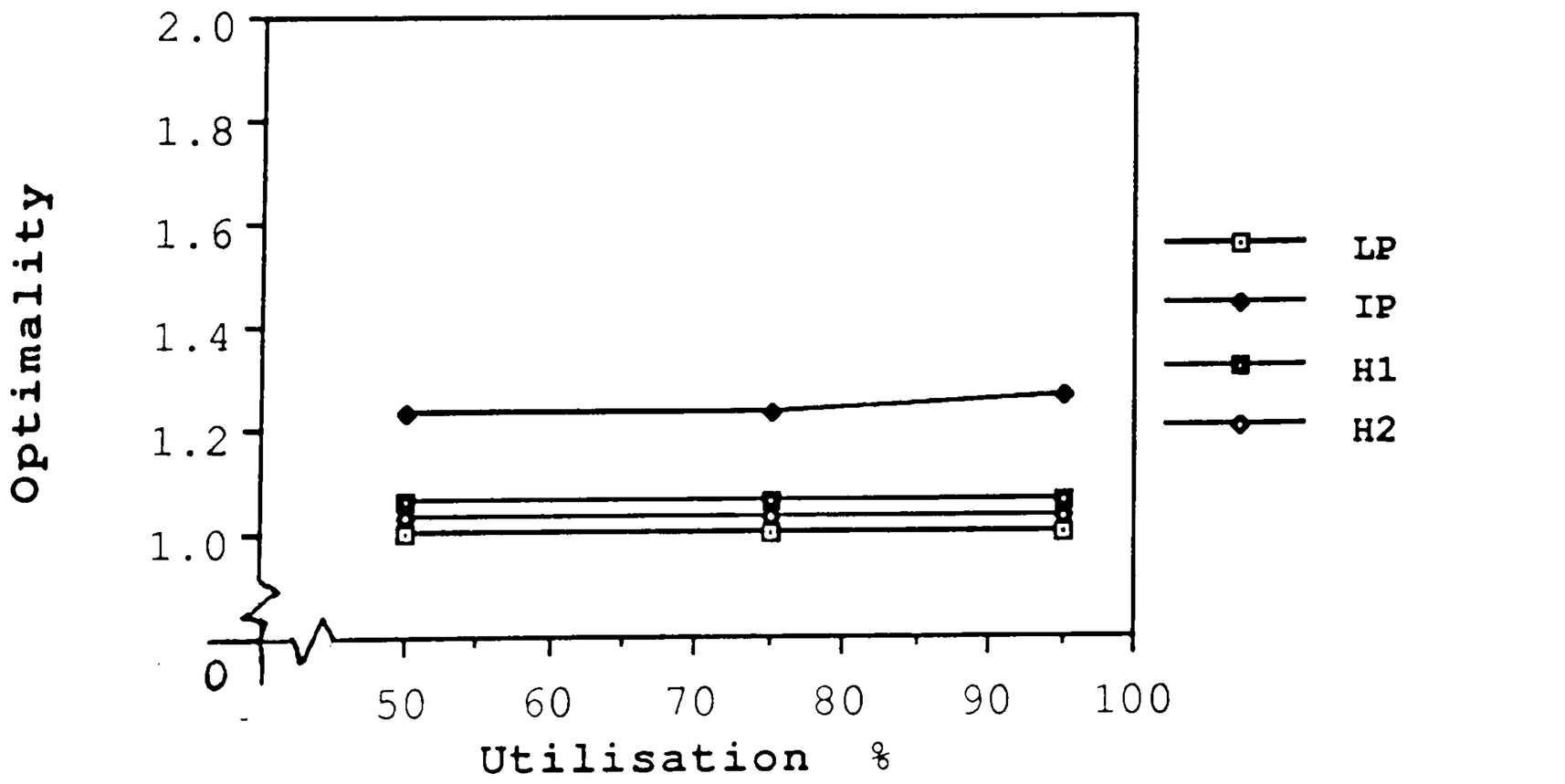




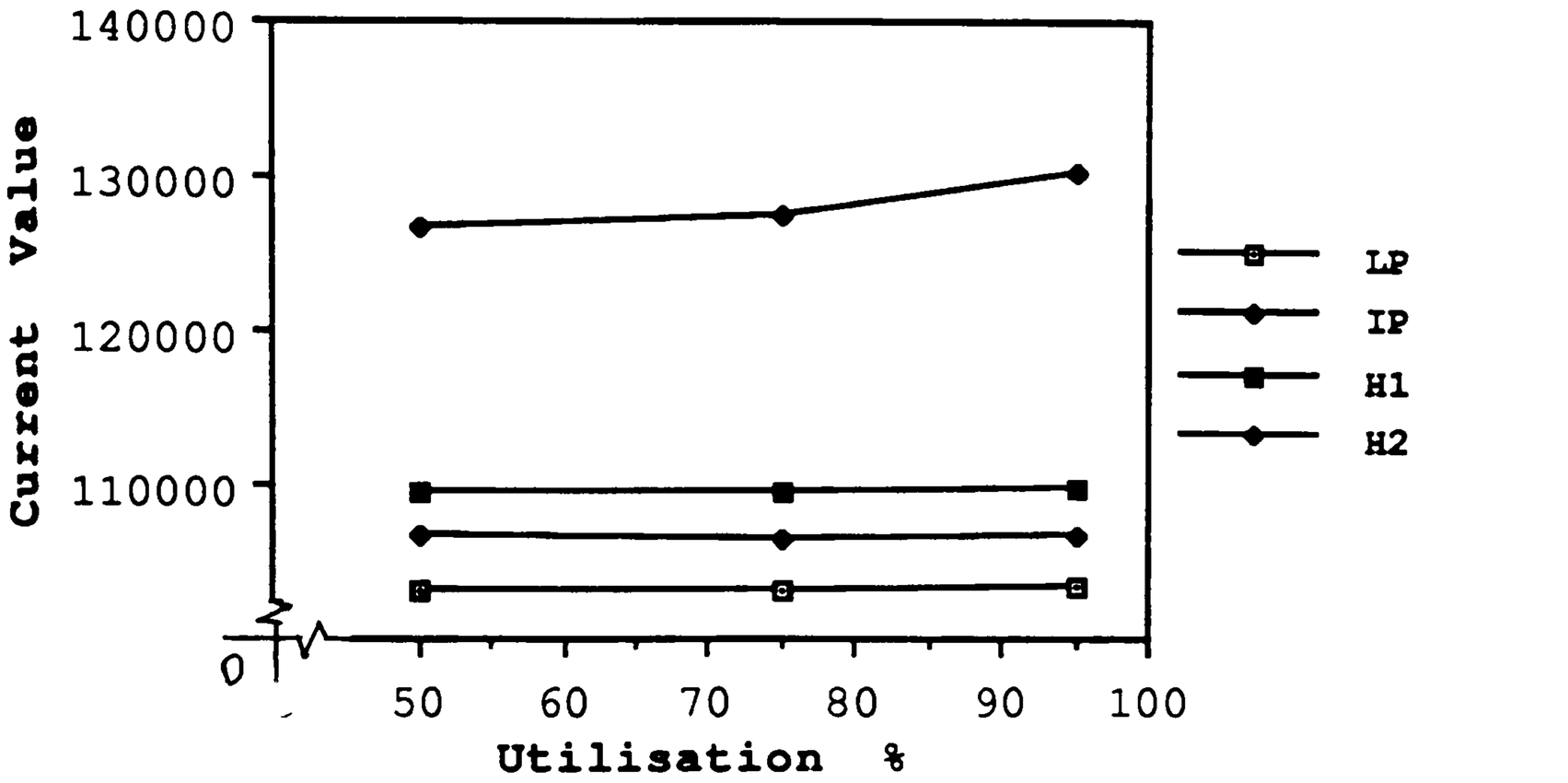
Graph 4.6.7.Five End Item Problem with Structure 1

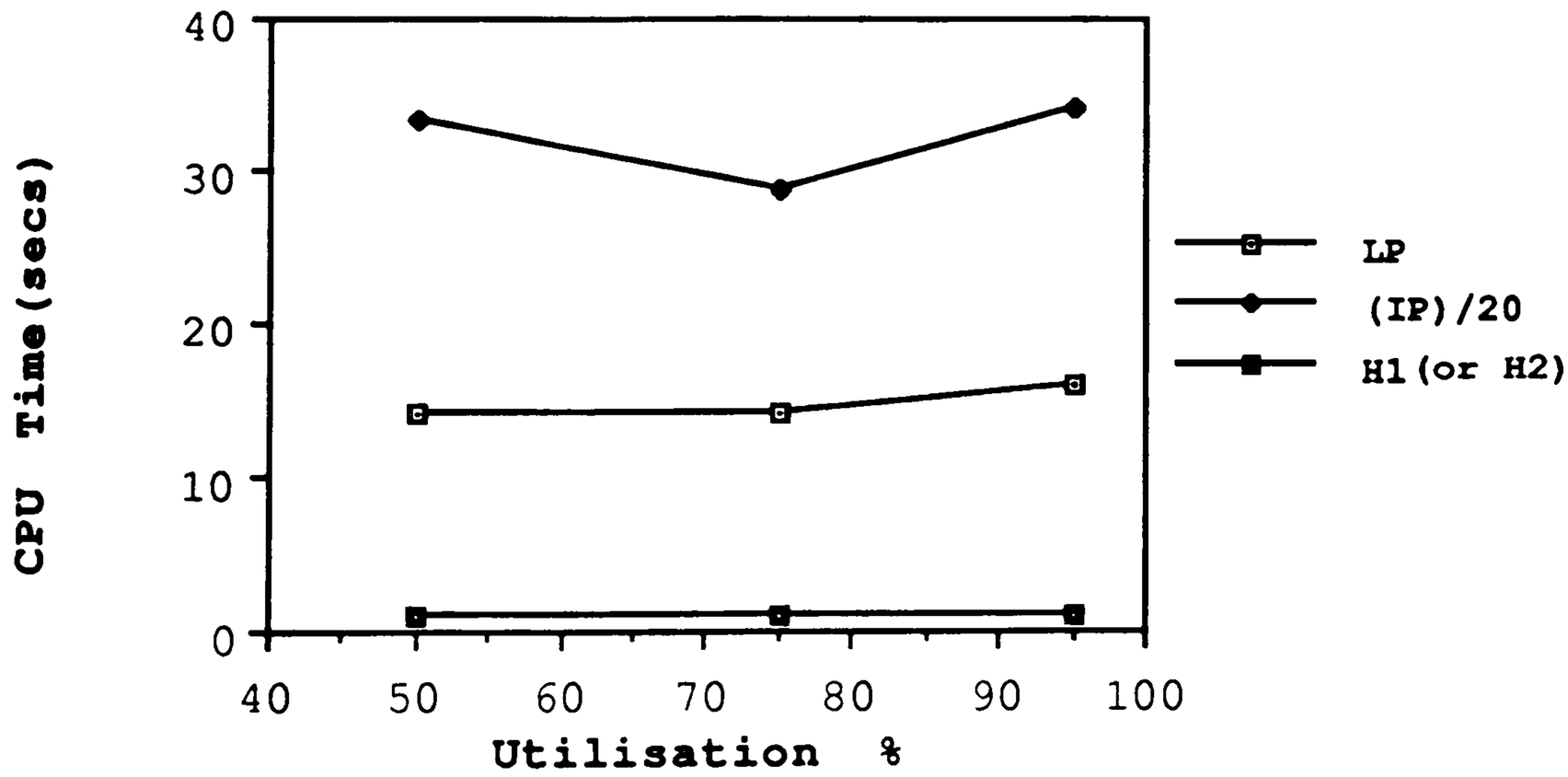


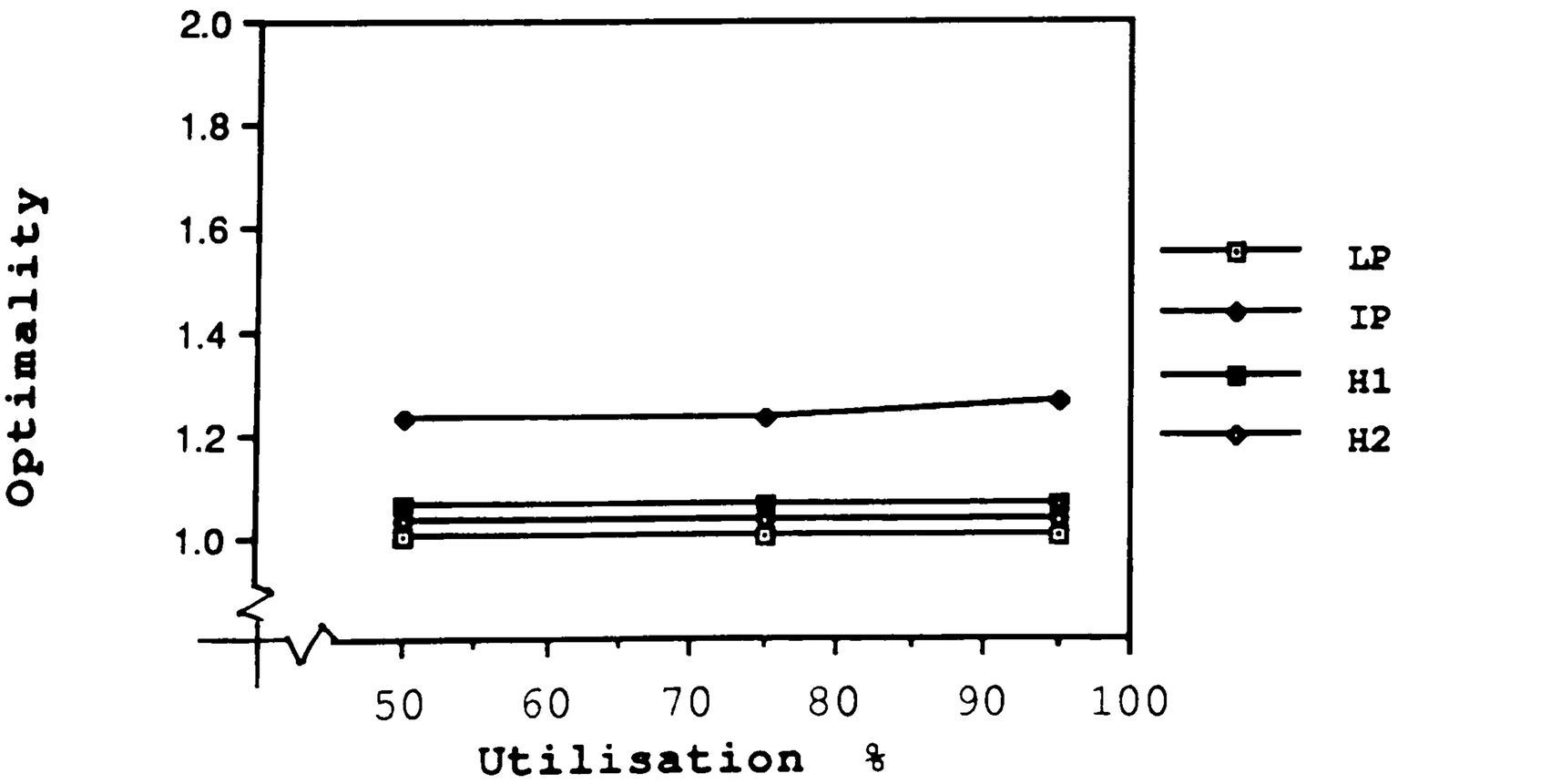




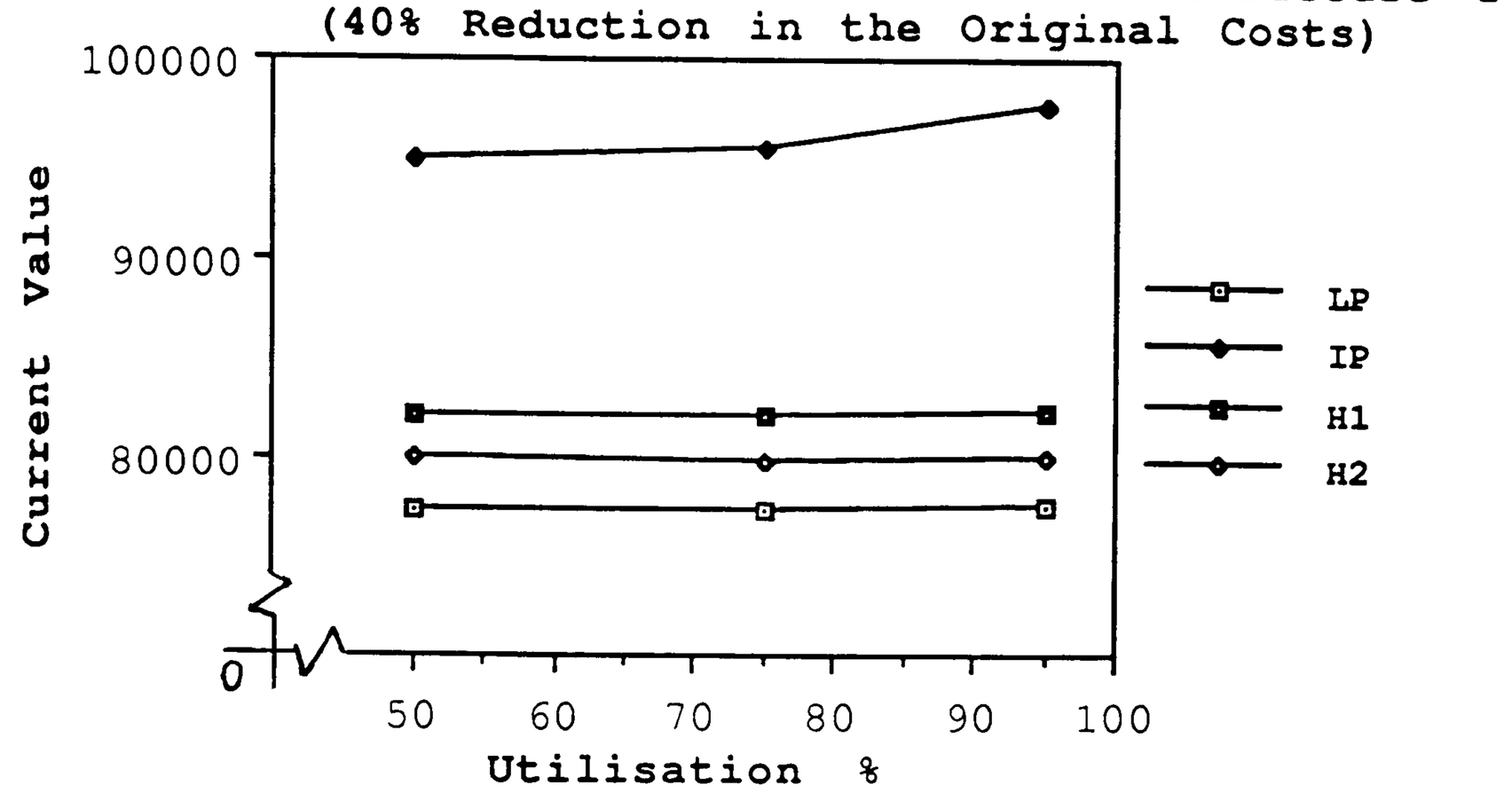
Graph 4.6.8.Five End Item Problem with Structure 1 (20% Reduction in the Original Costs)

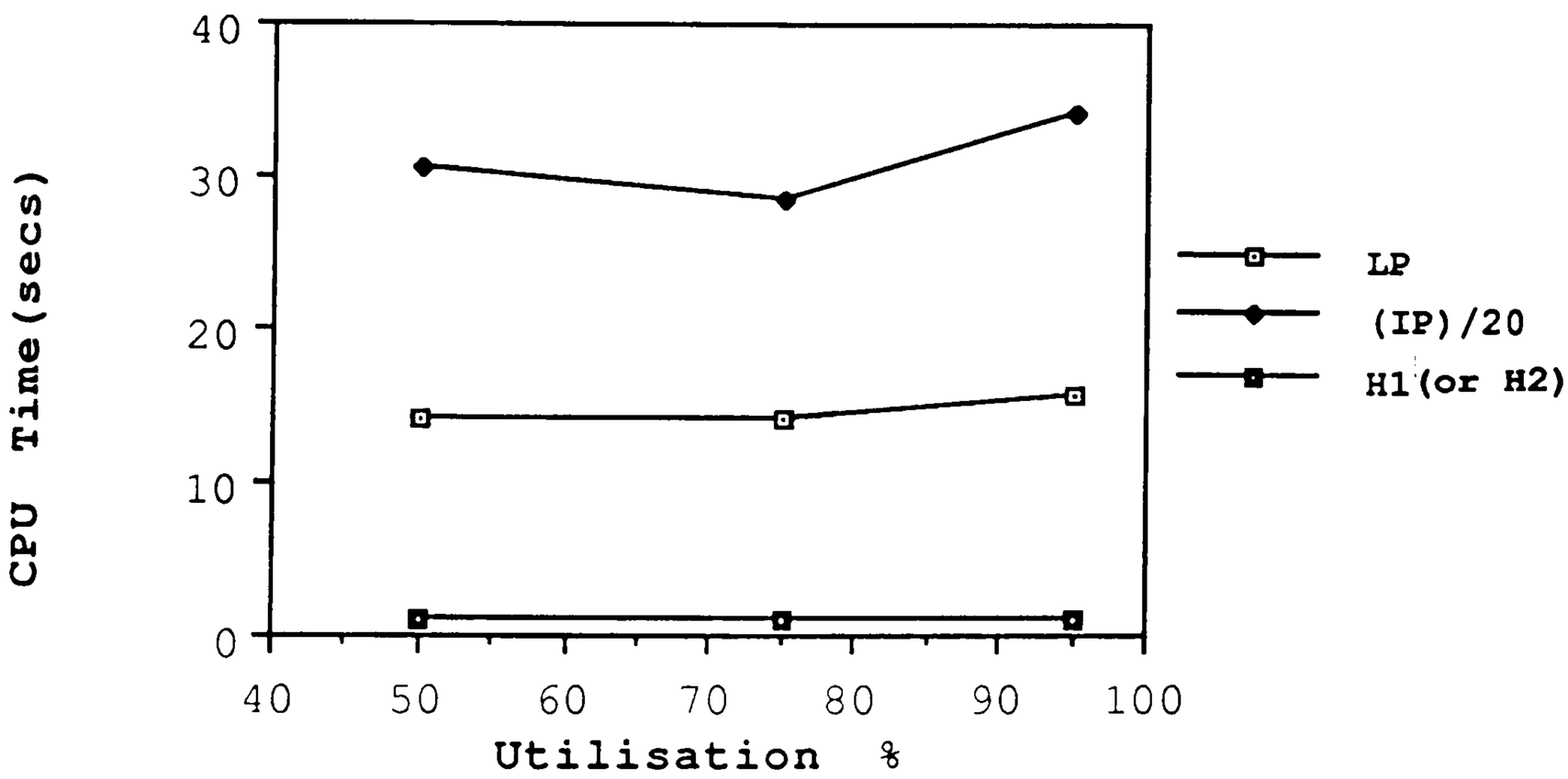


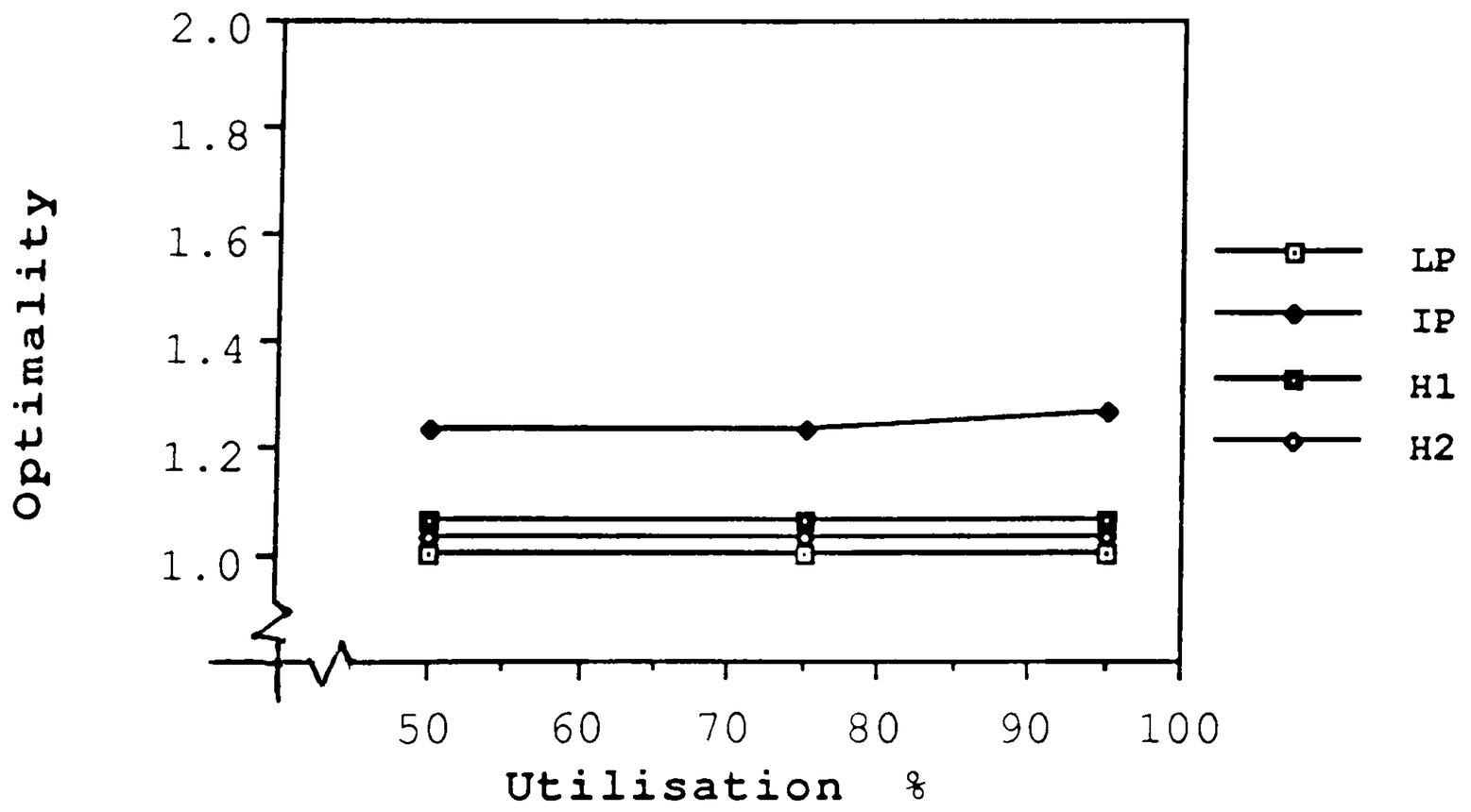




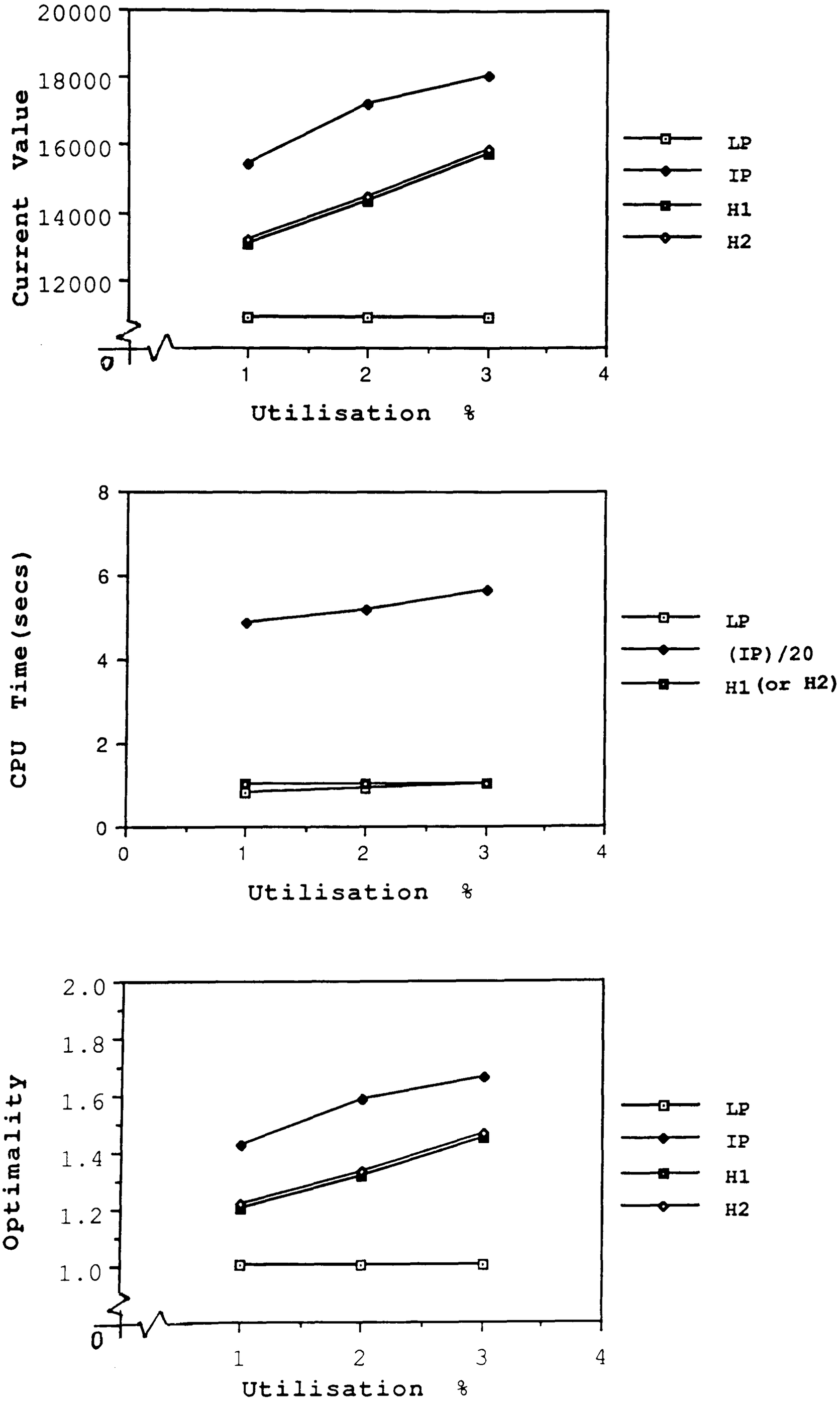
Graph 4.6.9.Five End Item Problem with Structure 1
(40% Reduction in the Original Costs)



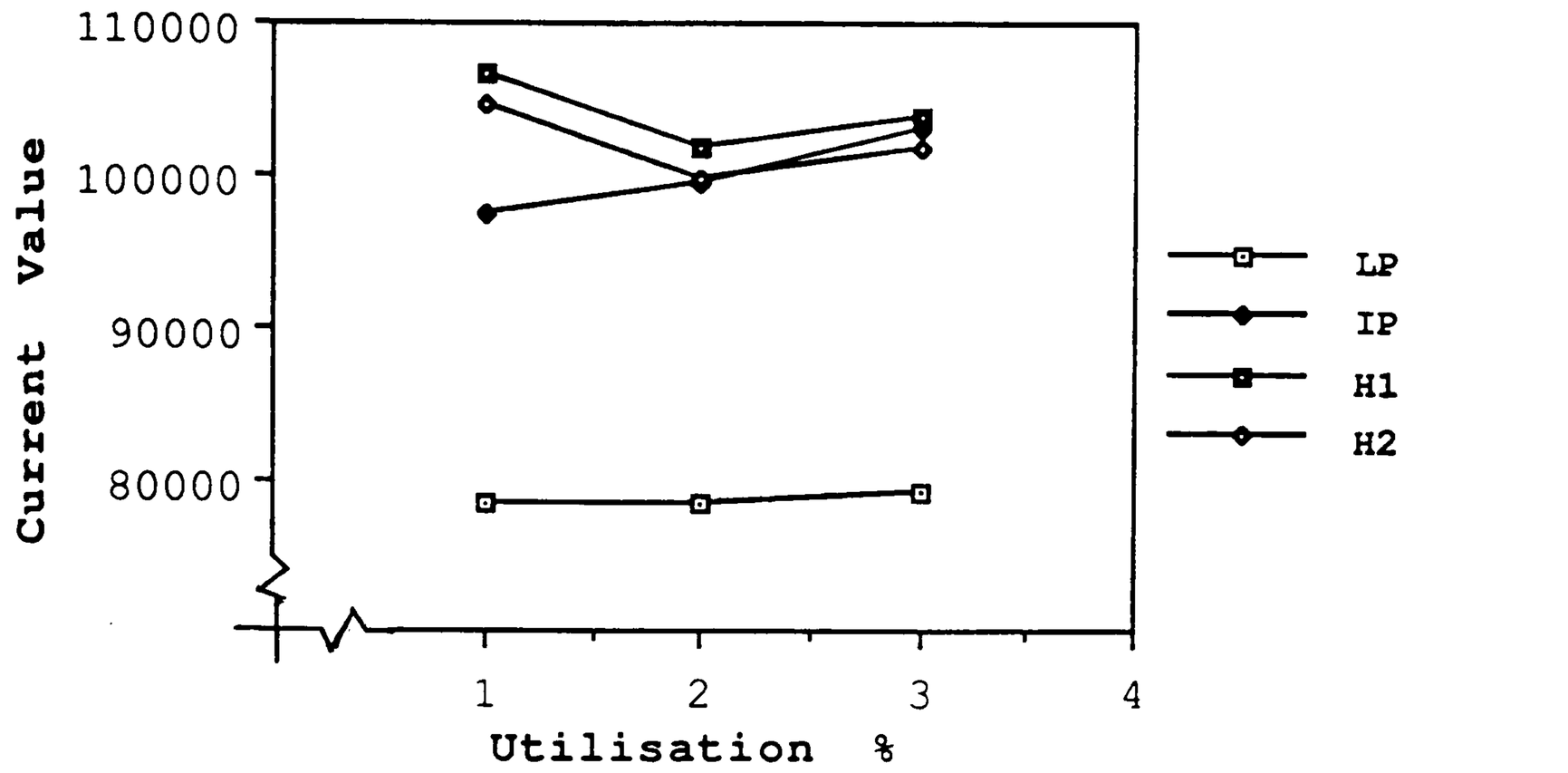


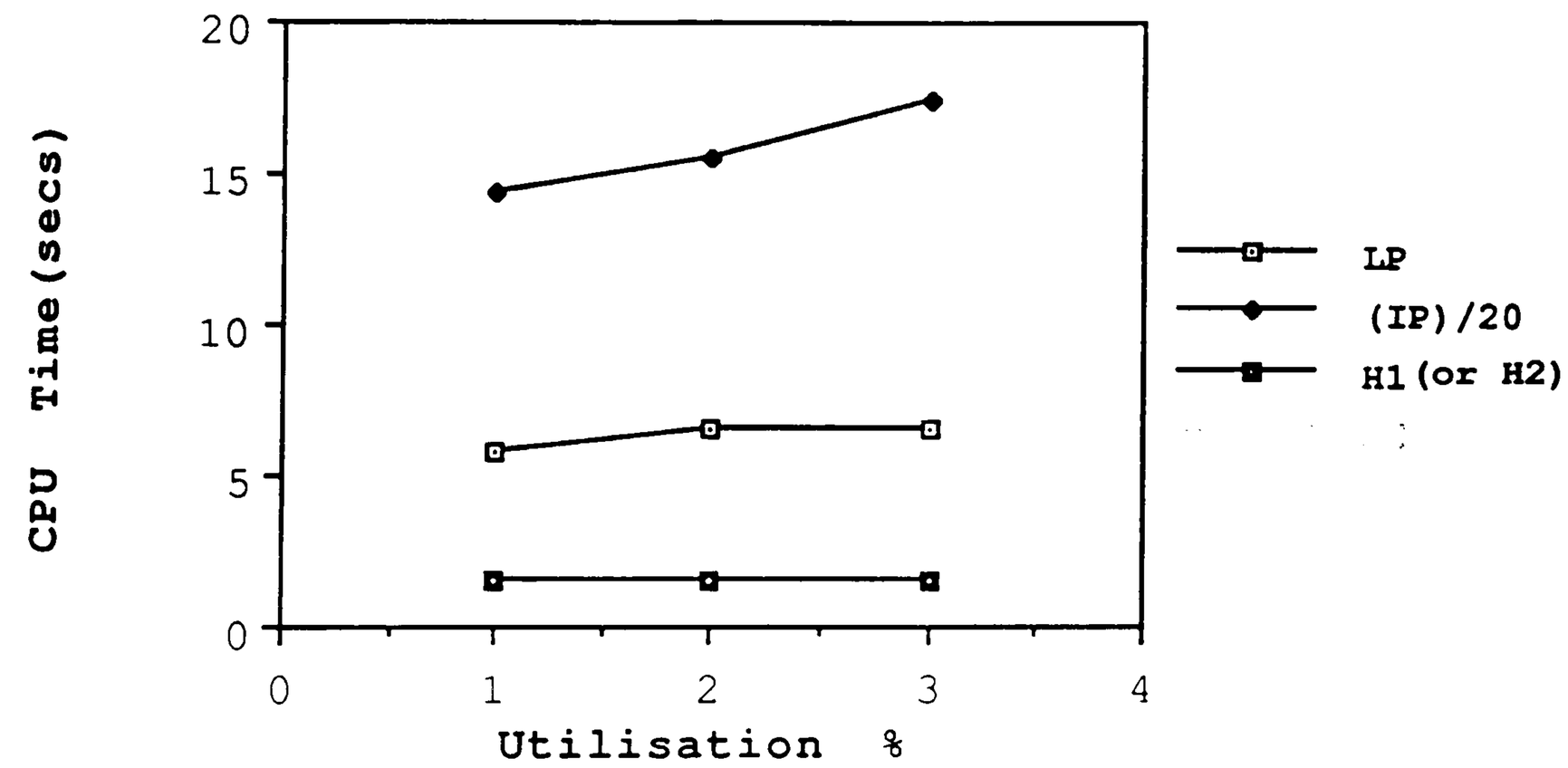


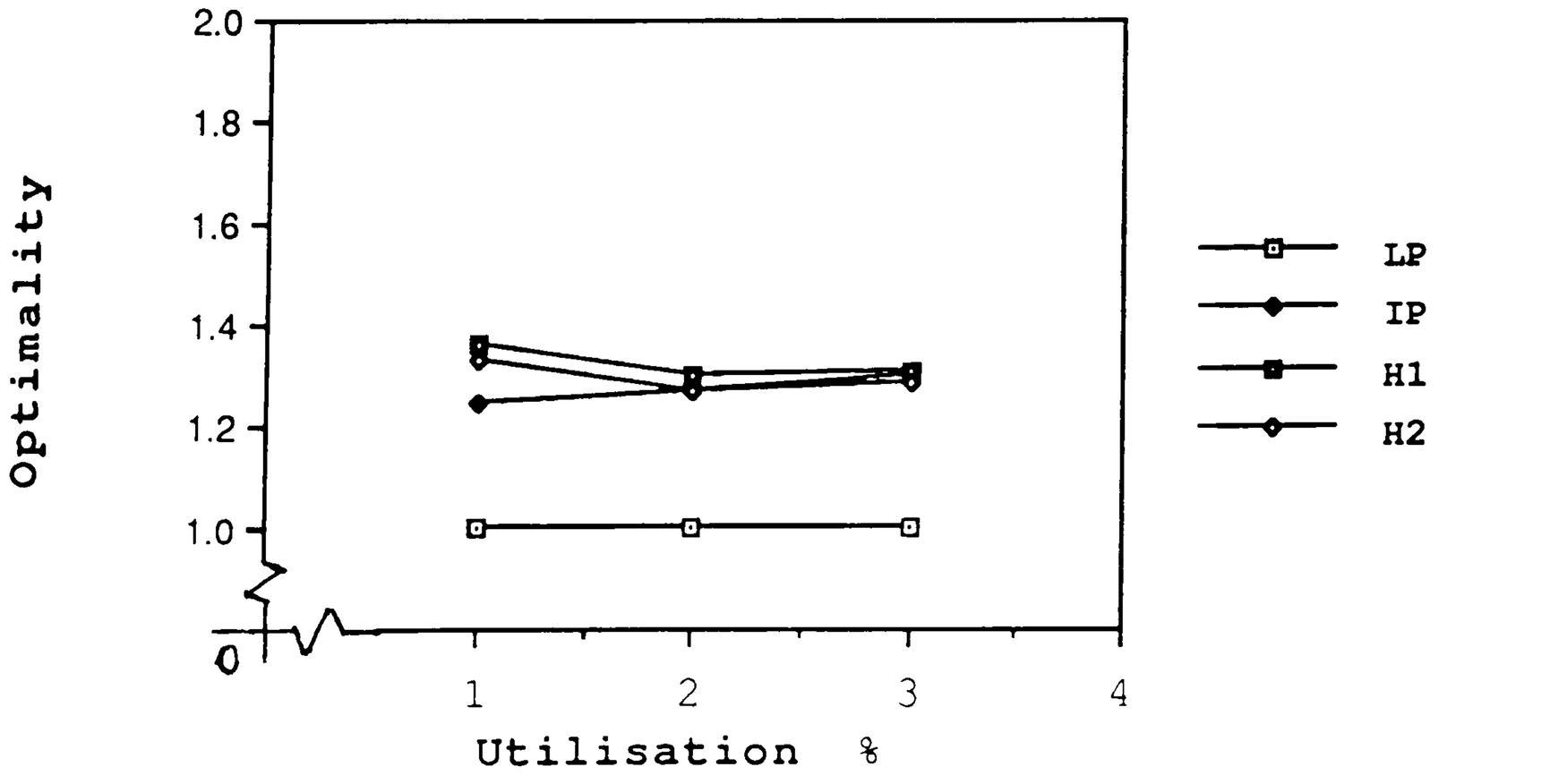
Graph 4.6.10. One End Item Problem with Structure 1



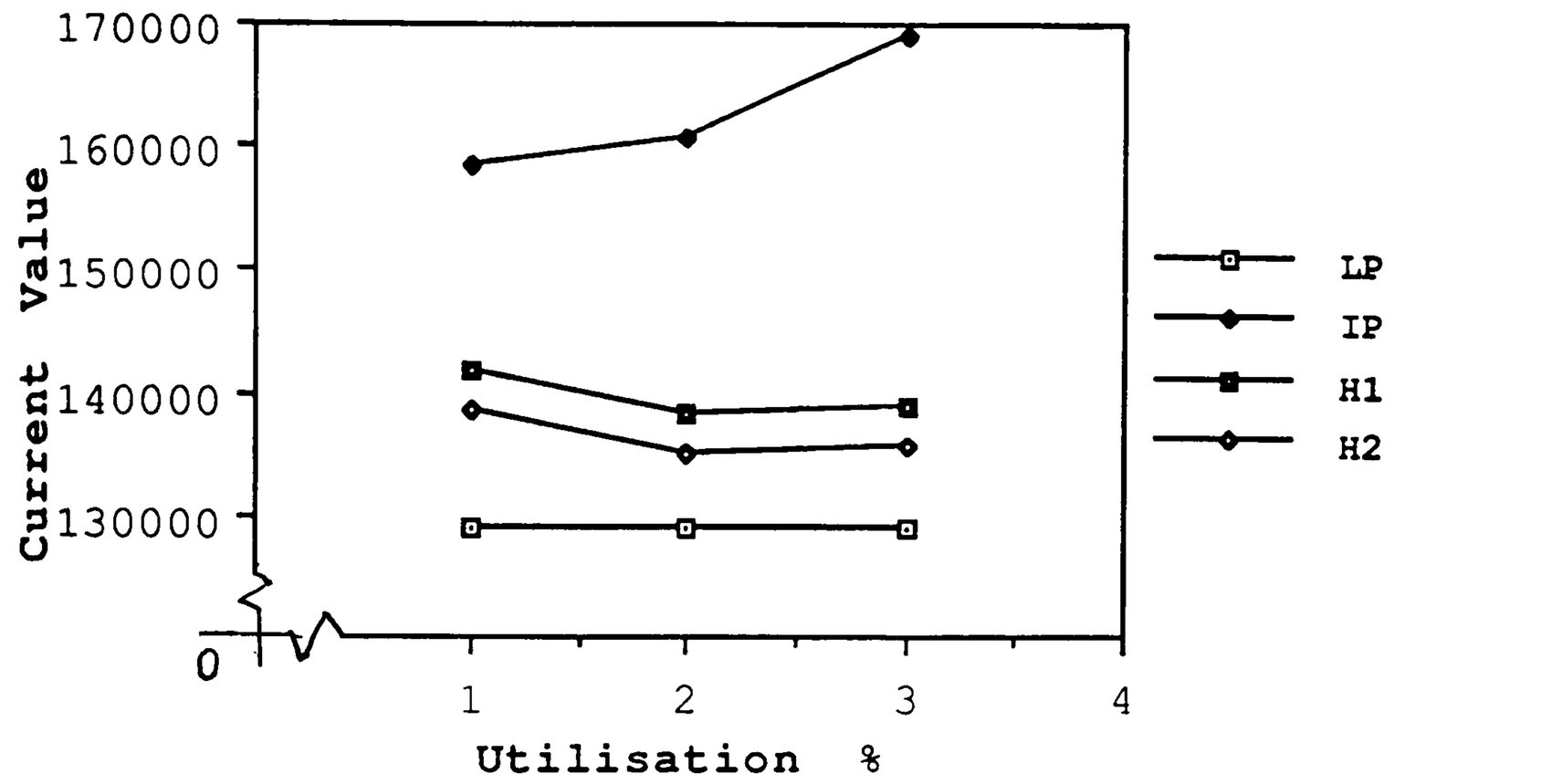
Graph 4.6.11.Three End Item Problem with Structure2

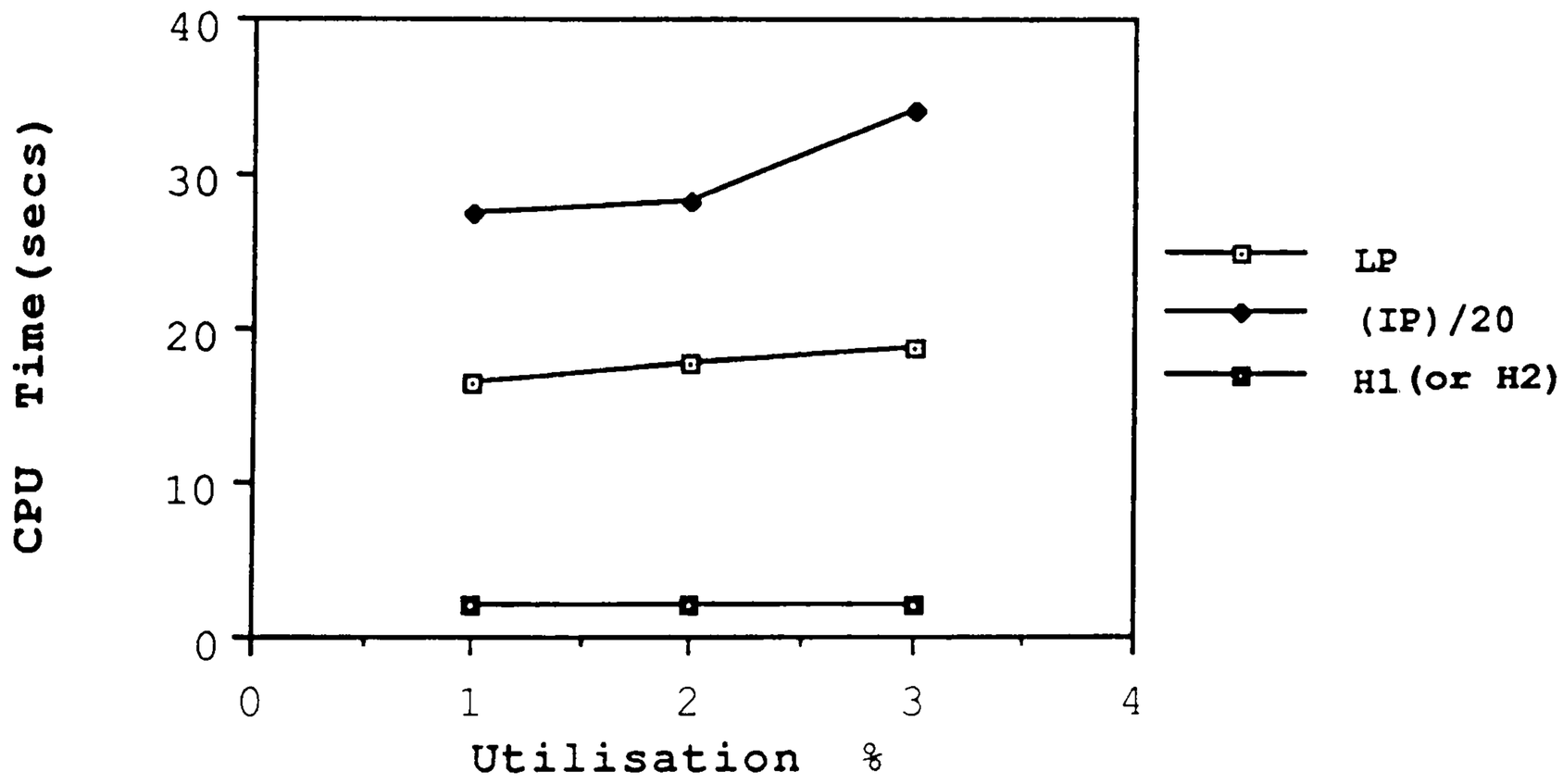


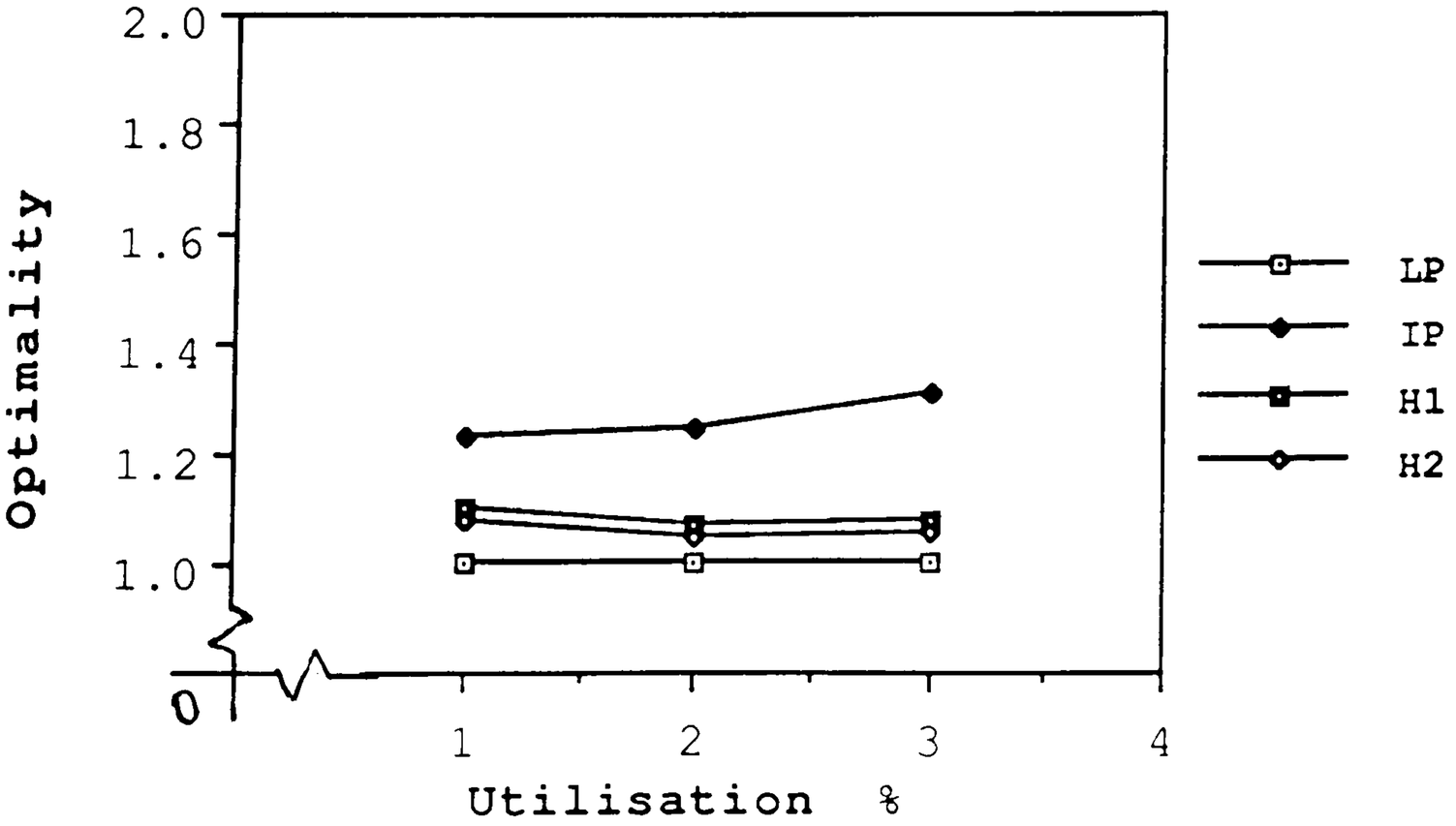




Graph 4.6.12.Five End Item Problem with Structure 3







CHAPTER 5

AN ANALYSIS OF MULTI-LEVEL LOT-SIZING PROBLEMS WITH BOTTLENECK UNDER A ROLLING SCHEDULE ENVIRONMENT

5.1. Introduction

Much research into multi-level lot-sizing has concentrated on fixed horizon problems, ignoring more realistic conditions (such as adding new demands into the planning period). The aim of this chapter is to illustrate that the heuristic, which was developed earlier in chapter 4, is applicable to the multi-level lot-sizing problem with a bottleneck under a rolling schedule environment. In this chapter, as was mentioned in chapter 2, the literature survey for the rolling schedule environment will be given in section 2, then the problem structure will be given in section 3, afterwards the simple heuristic for the problem with a rolling schedule will be given, and then an example problem will be analysed in section 7, and the final section will provide some conclusions.

5.2. Literature Survey for Rolling Schedule Environment

Lot-Sizing heuristics in the literature provide an optimal solution to lot-sizing problems under restricted assumptions in manufacturing firms, because most existing approaches assume that the fixed horizon for demand will remain the same without changes in the schedule. A fixed planning horizon will be used and each level will be treated independently. Although much progress has been made recently in multi-level lot-sizing problems, there

is still a gap between the problem faced by practitioners and models analysed by researchers. For instance, the problem of how lot-sizing decision rules would be used in real life problems has not been answered. To answer this question, a rolling schedule idea was developed in the past. The rolling schedule works according to the following process: a given multi-period problem is solved and the first decision is implemented. Then, the new information is appended to the planning horizon and the problem is solved again. This is repeated frequently. Thus, the rolling schedule is defined as a production plan for multi-period problems. However, in a rolling schedule environment, the addition of new information at the end of planning horizon may result in earlier decisions being altered or deleted and new orders added. This instability is referred to as nervousness in planned orders.

Carlson et al. [1979] were concerned with changed costs which occured when the production plan was changed to involve setups which had not been planned previously. They defined two sets of indices to decide whether there is a change in cost or not. These indices were: $A(x) = \{k \mid x_k = 0\}$, and $B(x) = \{k \mid x_k > 0\}$. According to these indices, they defined the new setup cost as follows:

$$w_k = \begin{cases} s_k + v_k & \text{for } k \in A(x) \\ s_k & \text{for } k \in B(x) \end{cases}$$

and including these new costs in the objective function gave:

$$C = \sum_{k=1}^{N} h_k \, I_{k+1} + \sum_{k=1}^{N} w_k \, \delta(x_k)$$

where

$$\delta(x_k) = \begin{cases} 1 & \text{if } x_k > 0 \\ 0 & \text{otherwise} \end{cases}$$

In this equation, x_k is production at period k, s_k is the setup cost at period k, v_k is the change cost for adding a new setup, $\delta(x_k)$ shows production indicator. They used the Wagner-Whitin algorithm to show how to reduce the nervousness.

Kropp and Carlson [1984] extended the Carlson et al. [1979] model by including a cancelled setup cost from setup to no setup situations. It means that the cancelled setup cost was involved in the model when the production of a particular period was greater than zero. They illustrated this by an index, B, where $B = \{k \mid x_k > 0\}$. They generalised the solution procedure which did not have any restriction on the change cost or setup cost. They defined the variable M as the first period where a setup was planned by $M = \min(k \mid x_k > 0)$. J was defined as the first period where net demand was positive after subtracting the initial inventory, where $J = \min(k \mid d_k > 0)$. If there was no net demand in periods, then the optimal solution to the problem was found to be $x_k=0$. They assume that the first setup in the plan will happen no later than period J, because they do not allow backorders. They defined the indices $L = \min(J, M-1)$, then the indices:

$C(x)=\{k \mid k \leq L \text{ and } x_k = 0\}$

$D(x)=\{k \mid k > L \text{ and } x_k = 0\}$

The objective of their research was to minimise the following objective function using the Wagner-Whitin lot-sizing heuristic:

$$\text{Total Cost} = \sum_{k=1}^{N} h_k \, I_{k+1} + \sum_{k=1}^{N} z_k \, \delta(x_k) + \sum_{k \varepsilon b(x)} u_k$$

subject to

$$z_k = \begin{cases} s_k - u_k & \text{for } k \varepsilon B(x) \\ s_k + v_k & \text{for } k \varepsilon C(x) \\ s_k & \text{for } k \varepsilon D(x) \end{cases}$$

where u_k is the cancelled setup cost.

Both Carlson et al. [1979] and Kropp and Carlson [1984] in their research used the Wagner-Whitin heuristic with the above formulations to reduce nervousness when applied to single-level lot-sizing problems without capacity considerations. The Wagner-Whitin heuristic illustrates more nervousness if it is compared with other heuristics such as the Silver-Meal, because the Wagner-Whitin heuristic is more horizon sensitive.

Baker and Peterson [1979] proposed an analytical structure for evaluating the cost performance of rolling schedules with quadratic cost functions. They focused on finding the relationship between the planning horizon and the cost performance efficiency which is the proportion of the cost of rolling schedules to that of an optimal solution. Their results illustrated that increasing the planning horizon results in improvements in performance but decreases returns. They also showed that the most important factor in their research was the cost structure which affects performance, but fluctuation and uncertainties in demand was a secondary factor.

Chand [1982] analysed the modified Wagner-Whitin algorithm and compared this algorithm with the Wagner-Whitin algorithm and Silver-Meal algorithm. He also proved that more information for the future periods is advantageous as the results of Baker [1979] showed. He illustrated that the modified Wagner-Whitin algorithm yields better cost performance than the others for the conditions of the single-level lot-sizing problem and unconstrained capacity.

Blackburn and Millen [1980] analysed the cost performance of rolling schedules using some of the well known heuristics such as Silver-Meal, Wagner-Whitin and the modified Silver-Meal. Under restricted assumptions such as a lumpy demand situation, the modified Silver-Meal heuristic performed better than the Silver-Meal or Wagner-Whitin heuristics for a short period horizon, but in a long planning horizon the modified Silver-Meal heuristic was not applicable due to lumpy demand. They showed that the Silver-Meal heuristic was superior to the Wagner-Whitin heuristic for long term horizons. The reason was that the Wagner-Whitin heuristic is horizon sensitive. The problems considered were single-level assembly systems and uncapacitated problems. Their later paper (Blackburn and Millen [1982]) exhibited a cost performance comparison between different lot-sizing techniques for the assembly systems under rolling schedules. They illustrated again that the Silver-Meal heuristic yields lower cost than the Wagner-Whitin heuristic and also required less computational time.

As Carlson et al. [1979] noted 'There is a feeling that although the optimality of the solution provided by the Wagner-Whitin algorithm is valuable, its price is too high: that price being the cost of changing plans. Many managers would rather live with non-optimal but stable plans.' Although many authors claim that their methods

produce an optimal solution in production scheduling, these are all subject to subobtimisation because they solve only the problems under certain assumptions.

The objective of this chapter is to illustrate that the heuristic, which was developed in chapter 4, for multi-level lot-sizing problems with a bottleneck is applicable to the rolling schedule environment for a parallel structure, which is a special case of MRP. This chapter also will compare the effect of the EOQ and Silver-Meal approaches under the rolling schedule environment. This comparison will be made by using the heuristic under conditions of normally distributed simulated demands.

5.3. Problem Structure

A capacity problem in the manufacturing firm occurs in different ways such as (1) incrementation of demand beyond the capacity system; (2) shortage of highly skilled operators; (3) scarcity of tools needed in any of the production stages. These situations result in a bottleneck problem which does not satisfy the external demand. A bottleneck is defined as a work centre which converts raw materials into finished items through the use of resources in the manufacturing environment. This chapter will concentrate on how much to produce from the limited resources, in which the capacity is not enough to satisfy the demands, for a rolling schedule.

The five stage product structure with a bottleneck problem illustrated in Figure 5.3 was used as the test bed for the heuristics. The five stages are illustrated vertically. The product structure, called a parallel structure in this chapter, is such that the assembly systems require each part (predecessor) to be the input to one successor stage. The five parallel production lines are illustrated horizontally and these end on the

product items, 1, 2, 3, 4, 5. These five-end-items are constrained by the bottleneck. In this product structure there is no commonality between stages as was assumed by Billington et al. [1986].

In section 5.7, the forecast window is stated as the future time periods in which demands are known. In the worked example, initially a problem which consists of demands in periods 1,...,6 was solved and the first decision for periods 1 to 3 was implemented. Then the procedure was rolled forward for the forecast window, periods 4 to 9.

For the results in Tables 5.8.1-5.8.9, the problem was solved for 96 time periods. Again, initially the problem was solved for periods 1 to 6 and the first decision for periods 1 to 3 was implemented. Then the procedure was repeatedly rolled forward for forecast windows from period k+3 to k+8 for k=1, 4, 7,..., 88.

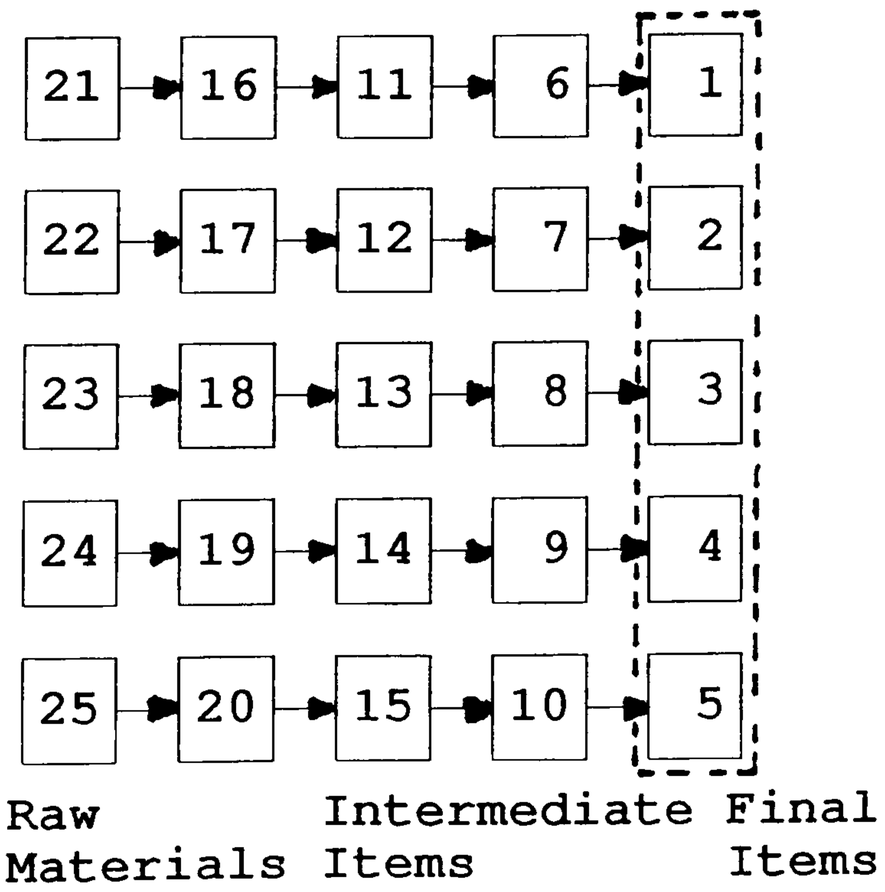


Fig.5.3. Five End Item Structure (the bottleneck is shown by dashed lines).

5.4. Assumptions

1. All lead times between stages are assumed to be zero,
2. Items which are produced by a firm are assumed independent,
3. Demand levels for the end items are known at a constant rate per year. In addition, there is no demand for the components at any intermediate stages. The number of predecessor components required at any intermediate stage is assumed to be equal to one. Again this follows the style of Billington et al. [1986]

5.5. Data Sets

A set of data was generated using a Normal Distribution to test the heuristic for a multi-level lot-sizing problem with a bottleneck(s) under the rolling schedule environment.

In the experimental studies three different cost structures, three demand streams and three capacity levels were used to generate the data using NAG subroutines [1990] on a Hewlett Packard system. For structure 1 of one-end-item problems, the average holding cost was set to 0.44 and the average setup cost was set to 400. For structures 2 and 3 the corresponding values were set to 1.00, 340 and 0.64, 420. Three demand streams were generated to give a low, medium and high coefficient of variation (C_v). C_v is calculated by dividing the standard deviation by the mean demand. Further details on the coefficient of variation will be given in chapter 6. The level of the coefficient of variation will be referred to as a structure. For example, a low coefficient of variation (C_v = 0.1057) will be referred to as structure 1, a medium coefficient of variation (C_v

= 0.1862) will be referred to as structure 2, and a high coefficient of variation (C_v = 0.3464) will be referred to as structure 3 for the one-end-item problem. The details of the other data sets are included in Appendix B. The capacity level is set after the demands are determined such that the total demand is divided by the product of the percentage of utilisation (which varies from 25% to 95%) and the number of time periods. All these data are in the style of Billington et al. [1986]. To aid explanation, the solution to a small problem is illustrated in the example section.

5.6. Simple Heuristic for the Problem with Rolling Schedule

The heuristic starts by first dividing the product items into end-items and non-end-items. Because each non-end-item's production is unconstrained, and non-end-items or end-items with bottleneck are constrained by the bottleneck so that they do not affect the production of other items. A succesful development in the solution of the multi-level lot-sizing problem was reported in chapter 4. In our example problem the bottleneck is always limited to effectively the end-items. This does not mean that no bottleneck could occur in other product item(s). The heuristic illustrates that two simple procedures can be used independently to find the production levels if the production items group into (a) items constrained by the bottleneck; and (b) items which are unconstrained.

5.6.1. Non-End-Items with Bottleneck

If the demand for end-items is known in advance, it does not invalidate the independence of the production of intermediate or raw material items. These capacitated product items must share the resources of the bottleneck

so that the determination of when to produce end-items is more complex. Because of this reason the production problem is a constrained problem. To deal with this kind of problem a simple heuristic was adopted that would adopt a greedy approach to production with heavy utilisation of production capacity. In addition, the heuristic operates in a cyclic manner, moving the product items in turn, to produce reasonably smooth production. In chapter 4, this heuristic is shown in detail. Two cases will be given to demonstrate the heuristic and one case will be examined with an example problem in the next section showing the cylical manner.

Case (a) 3-end-item problem

Using the heuristic which is developed in chapter 4 the approach is to produce as much of product item 1 in the first period as is equal to the bottleneck's capacity less the demand for product items 2 and 3 that would satisfy their demand for several periods. In period 2, assuming there is a sufficient stock for product item 1, produce as much of product item 2 as is equal to the bottleneck's capacity less the demand for product item 3. In period 3, assuming the stocks of product items 1 and 2 are enough to satisfy demands, produce as much of product item 3 as is equal to the bottleneck's capacity and so on. (See also case (b) in section 4.4.2).

Whenever any of the stock levels of product item would become negative, that particular product item is produced again until the production of that product item becomes dominant over the next one or more periods.

The priority allocation chosen as the basis for the results in this thesis, was used by Billington et al.[1986]. No particular priority allocation was suggested by the data itself, and alternative priority

allocations to that above, which were tested intensively on the basis of the following criteria (i) largest total demand first, and (ii) most variable demand first, did not change the results significantly. It has therefore been assumed that choosing an alternative priority allocation, to the one chosen in this thesis would not make a significant difference to the results. However, the comparisons of the alternative priority allocations would be an interesting topic for further research.

Case (b) 5-end-item problem

For this problem, in order to keep the heuristic simple, product items are grouped in such a way as to keep identical items together eg product items (1,2,3) and (4,5) respectively. These two sets of product are now treated as single products and a three period cycle (in case a) was modified to a two period cycle. This grouping into two sets was chosen, rather than other alternative sets, in order to satisfy the two criteria which are explained in chapter 4 (case (b)).

The total cost is now the sum of the individual combined inventory and setup costs for the end-items-and non-end-items.

5.7. An Example

A small One-End-Item problem will now be considered. In the problem five holding costs (0.5,0.1,0.5,0.1,1.0) and five setup costs (300,200,200,500,400), and also six period demands (38,43,37,42,33,42) were used. The solutions are shown for the end item and components in turn:

5.7.1. Rolling schedule on bottleneck items

The heuristic was to produce as much of product item i as is equal to the bottleneck capacity utilisation, in order that there will be more than enough for several periods.

Let the bottleneck be located such that it affects the end item, the heuristic which will be used is that $P_{it} = cap_t$ where P_{it} is the units of production item i in period t, and cap_t is the bottleneck capacity utilisation, which is 50% in this case, at work centre at time t, then produce up to $S_{it} < d_{it}$ where S_{it} is stock of product item i in period t and d_{it} is the demand for product item i during period t. So from the above demand patterns the capacity utilisation $cap_t = 79$ was found. So the original schedule will be:

Period	1	2	3	4	5	6
Demand	38	43	37	42	33	42
P_{it}	79	79	0	77	0	0

The optimal solution to this problem is $P_{11} = 79$, $P_{12} = 79$, $P_{14} = 77$, and the rest of the periods for product item 1 are zero. It is assumed that there is a set of demands at the end of period 3, such that $d_{14} = 42$, $d_{15} = 33$, $d_{16} = 42$, $d_{17} = 40$, $d_{18} = 48$, $d_{19} = 38$. These demands will lead to the new set of results.

Period	4	5	6	7	8	9
Demand	42	33	42	40	48	38
P_{it}	81	0	0	81	41	0

As this schedule illustrates, the original plan has changed. Production was 77 in period 4, but appending the new set of demands at the end of period 3 in the planning horizon results in 81 instead of the original 77. For the calculation of the total cost for these problems for the bottleneck facility, it is necessary to bear in mind

that the revised total cost now includes the original cost up to the beginning of the new schedule.

The objective function to calculate the total cost is adopted from Billington et al. [1986] and from now on whenever the total cost is mentioned, it will refer to this equation which is illustrated below.

$$TC = \sum_{t=1}^{T} h_i*(T - t + 1)*P_{it} + cs_i*X_{it}$$

The total cost for the original plan is 1420, and the total cost for the revised schedule is 2340.

5.7.2. Rolling schedule on non-bottleneck items

Two well known approaches to cope with relatively varying demand will be used on the non-bottleneck product items. They are EOQ and Silver-Meal approaches.

The application of the classic EOQ approach is very difficult for real life problems. Because of that the assumptions are relaxed to get closer to the real circumstances; ie, different holding and setup costs for different items, demands for the production items are relatively deterministic but changes during the time horizon. This is the condition of MRP application.

5.7.2.1. The Economic Order Quantity Approach (EOQ)

Consider product item 3 which is useful for demonstrating the application of the EOQ approach. The same demands will be used for product item 3, because only one predecessor item is required to produce the successor item in the product structure. The EOQ is 177, which is

less than the total demand, so that there are three alternatives that will apply.

Alternative a: Produce Q_i which is the Economic Order Quantity for product item i, based on setup cost cs_i and holding cost h_i, in period 1 for one or more period to satisfy the demand for product item i. Before the stock drops below zero, produce another Q_i for another period or periods. Continue this process t_r times where

$$t_r=[\sum_{t=1}^{T} d_{it} / Q_i + 0.99]$$ which is 2 times. The value of 0.99 is included here in order to round up to the next integer above.

Period	1	2	3	4	5	6
Demand	38	43	37	42	33	42
P_{it}	177	0	0	0	58	0

The total cost for alternative a is 989.

Alternative b; Produce Z_r items in period 1 and then next produce Z_r when stocks would become negative which means that no production is made (i.e. $S_{in} < d_{in}$)

$$Z_r = \sum_{t=1}^{T} d_{it} / t_r$$ which is 118 or 117.

So the schedule will be

Period	1	2	3	4	5	6
Demand	38	43	37	42	33	42
P_{it}	118	0	0	117	0	0

The total cost for alternative b is 930.

Alternative c; Produce all demands in period 1

Periods	1	2	3	4	5	6
Demand	38	43	37	42	33	42
P_{it}	235	0	0	0	0	0

The total cost for alternative c is 905.

To start rolling forward, the minimum scheduled cost which is given in alternative c is chosen. Assume that some future demand has been received in period 4, the schedule is changed as below. Then P_{i7} is 126.

Period	4	5	6	7	8	9
Demand	42	33	42	40	48	38
P_{it}	0	0	0	126	0	0

Then the total cost of rolling forward is 1294.

5.7.2.2. The Silver-Meal Approach

In this section, one of the well known heuristics, the Silver-Meal [1973], will be examined for comparison with the EOQ approach for the non-bottleneck items. This heuristic selects the lot-sizes in order to minimise the total cost over the planning horizon under the varying demand conditions. The heuristic, which was explained in chapter 4, was:

Let T equal the number of periods,

$$\text{Average Cost (AC)} = \frac{\text{Setup Cost + Total Holding Cost}}{T}$$

or

$$AC = \frac{cs_i + [(1-1)*d_{i1} + (2-1)*d_{i2} + \ldots + (T-1)*d_{it}][h_{it}]}{T}$$

Note that there is no inventory carrying cost during the first period so that d_{i1} drops from the formulation. When the total cost for a particular period is less than the succeeding one, the production period (T) is found and the quantity of the lot-size is the sum of the demands,

which is the $Q_{it} = d_{i1} + d_{i2} +...+ d_{it}$ during the period T. This process continues through all periods. For the same problem, using the Silver-Meal heuristic, according to the procedure above, the schedule will be;

Period	1	2	3	4	5	6
Demands	38	43	37	42	33	42
P_{it}	193	0	0	0	0	42

Total cost for the original plan is 1000.

If the plan is rolled forward to ninth period, the plan will be

Period	4	5	6	7	8	9
Demand	42	33	42	40	48	38
P_{it}	0	0	168	0	0	0

Total cost for the new plan is 1315.

As will be seen from the solution of the original problem the new schedule costs are generally increased under the rolling schedule environment. In this case in the original schedule, production was 42 in period 6 but according to the new demand production should be 168 in period 6. For another product item, the procedure will be the same.

5.8. The Results of Rolling Schedule

A program was written in Fortran 77 for the multi-level lot-sizing problem under the rolling schedule when there was a bottleneck(s). Normally distributed random numbers were generated using a NAG subroutine (included in Appendix D).

It was assumed that there was either one bottleneck which occurred in the final product item(s), or two bottlenecks which occurred, in the final product item(s), and the

intermediate product item(s) respectively (i.e. the final product item 1 and intermediate product item 3 for the one-end-item problems). Then, the heuristic was applied to the constrained (bottleneck) product items while the EOQ and Silver-Meal were employed on the unconstrained (non bottleneck) product items for the rolling schedule.

The results with different capacity utilisations are illustrated in Tables 5.8.1-5.8.9. In each table, the total cost of the problems using EOQ and Silver-Meal for the unconstrained product items with the heuristic for the constrained product items are illustrated. The second and third columns refer to problems without a rolling schedule and the fourth and fifth columns refer to problems with a rolling schedule. The multiple bottleneck results are shown in the first row, and single bottleneck cases with different work centre utilisation, i.e. 50% and 75%, are illustrated in the second and third rows, respectively. The same results as presented in the Tables are also presented in Graphs 5.8.1-5.8.9. In each graph, the number 1 refers to the multiple bottleneck cases (where the bottlenecks occurred in the final product item with 75% capacity utilisation and the intermediate product item with 50% capacity utilisation). Likewise, the numbers 2 and 3 refer to single bottleneck cases where the bottlenecks always occurred at the final product items with 50% and 75% utilisation. The total costs are shown on the vertical axis. Furthermore, the legends H1, H2 refer to the total cost of problems using EOQ and Silver-Meal for the unconstrained product items and the heuristic for the constrained product items without rolling schedule while the legends H3, H4 refer to the solution using EOQ and Silver-Meal for the unconstrained product items and the heuristic for the constrained product items with rolling schedule, respectively. According to the results, the total cost is increased when appending new demands in the planning

horizon. The results also illustrate that the heuristic with Silver-Meal is better than the heuristic with EOQ because Silver-Meal performs well with varying demand. More discussion will be given in chapter 6.

5.9. Conclusion

It was shown that the dynamic problem can be handled using the heuristic, which was developed in chapter 4, for constrained product items and two well known approaches (EOQ and Silver-Meal) for the unconstrained product items to make stable plans.

Table 5.8.1. One End Item Problem with Structure One

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	177047	158000	236019	235485
50% Util.	159790	132834	207368	206567
75% Util.	177121	150165	228002	227201

Table 5.8.2. One End Item Problem with Structure Two

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	313810	292097	450704	438278
50% Util.	285904	250619	424394	399542
75% Util.	315850	280566	459203	434351

Table 5.8.3. One End Item Problem with Structure Three

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	322818	263433	443883	406937
50% Util.	262283	201299	363994	327048
75% Util.	280650	219667	385354	348408

Table 5.8.4. Three End Item Problem with Structure One

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	1065813	937737	1330438	1286940
50% Util.	1042958	859223	1376117	1309539
75% Util.	1070519	886784	1396423	1329845

Table 5.8.5. Three End Item Problem with Structure Two

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	1442978	1275779	1832941	1750423
50% Util.	1424898	1184607	1882821	1800479
75% Util.	1436365	1196073	1892480	1810137

Table 5.8.6. Three End Item Problem with Structure Three

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ.	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	1020687	910895	1289637	1203334
50% Util.	959582	808717	1291359	1203364
75% Util.	968396	817531	1295945	1207950

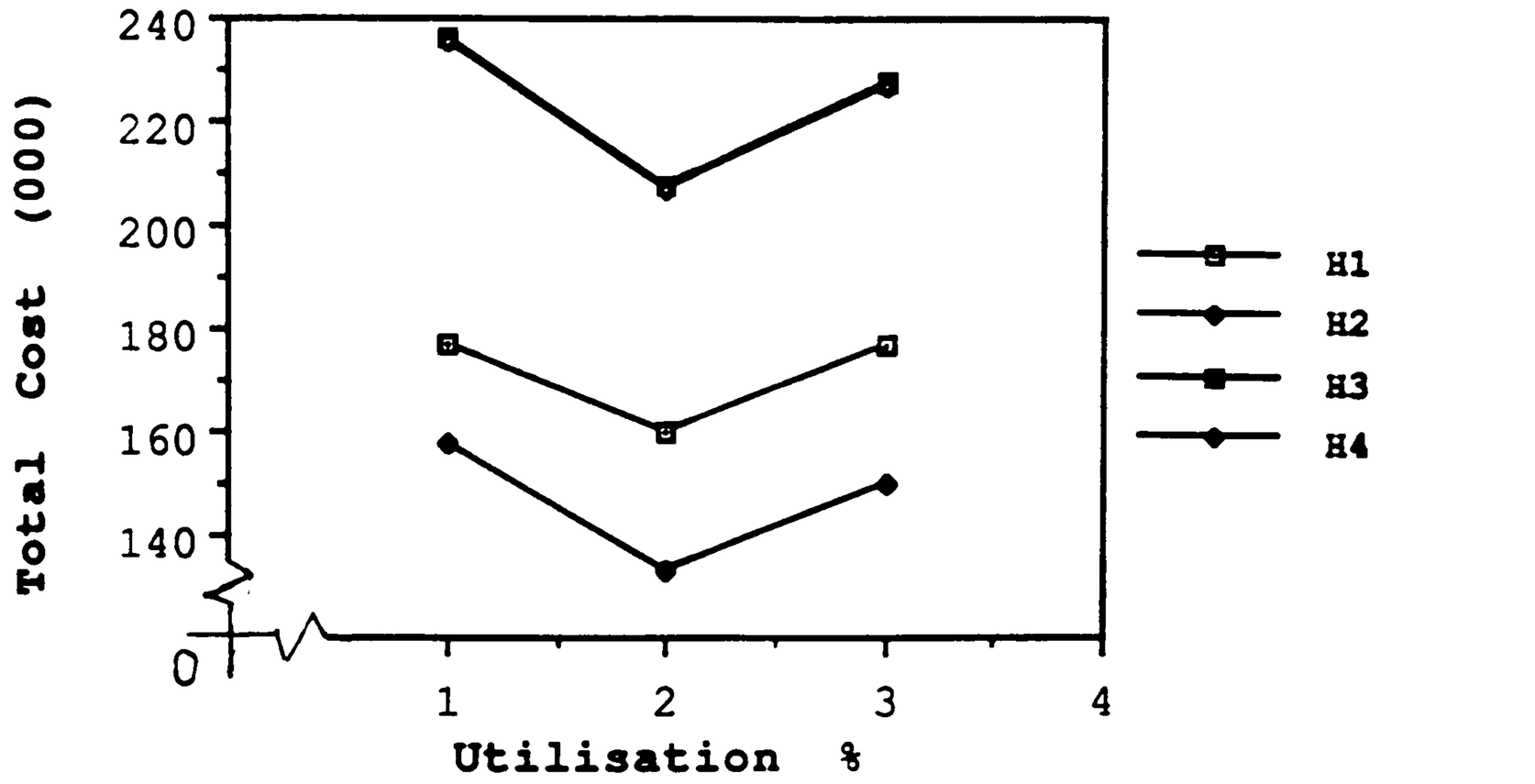
Table 5.8.7. Five End Item Problem with Structure One

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	1196459	1053393	1581905	1564095
50% Util.	1149856	982858	1541031	1516214
75% Util.	1141588	974590	1554452	1529695

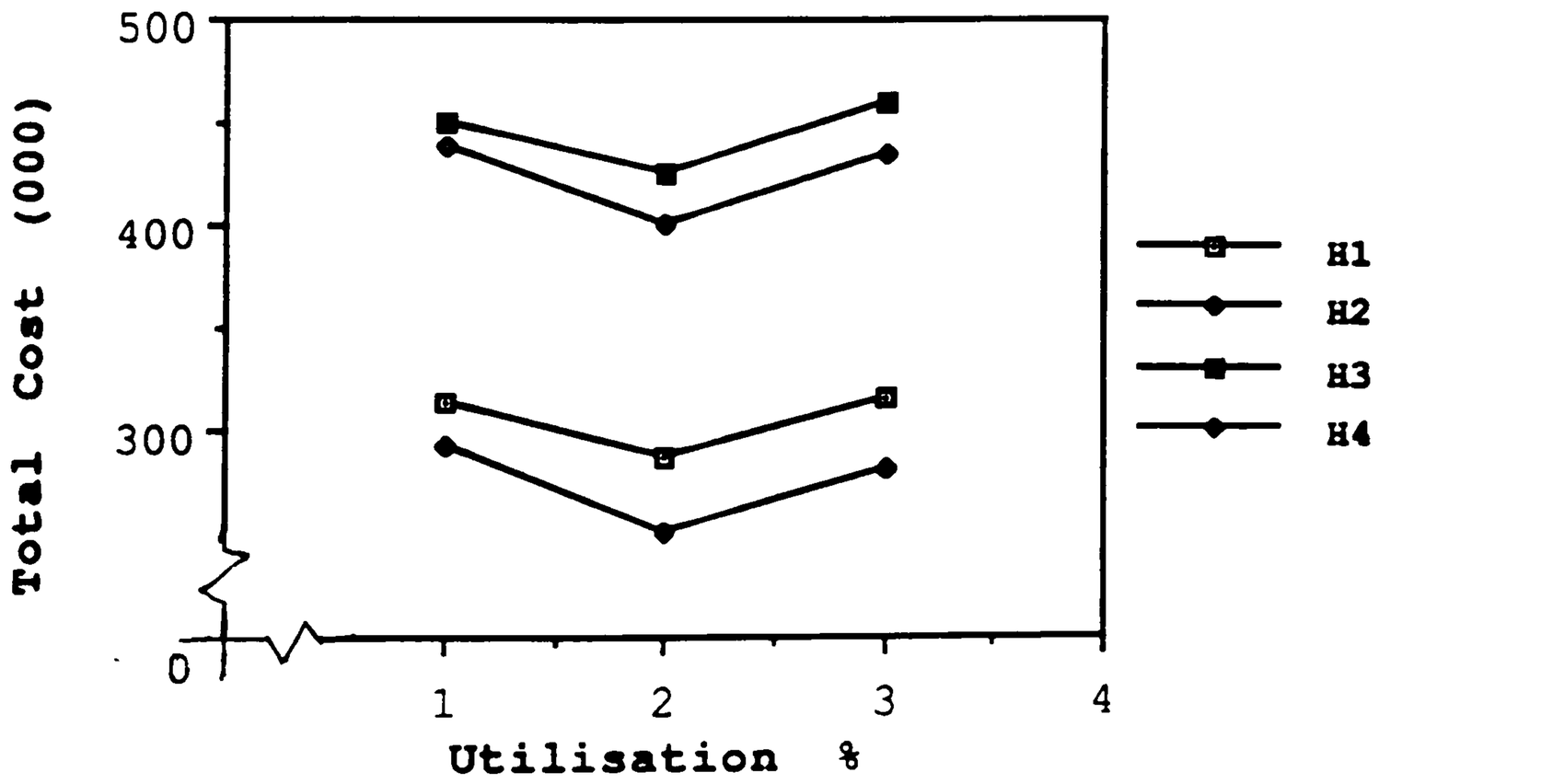
Table 5.8.8. Five End Item Problem with Structure Two

	Heuristic Solution			
	Without Rolling Schedule		With Rolling Schedule	
	EOQ	Silver-Meal	EOQ	Silver-Meal
50%+ 75% Util.	1102177	968298	1478366	1441051
50% Util.	1047246	876317	1440604	1391606
75% Util.	1062284	891355	1467602	1418604

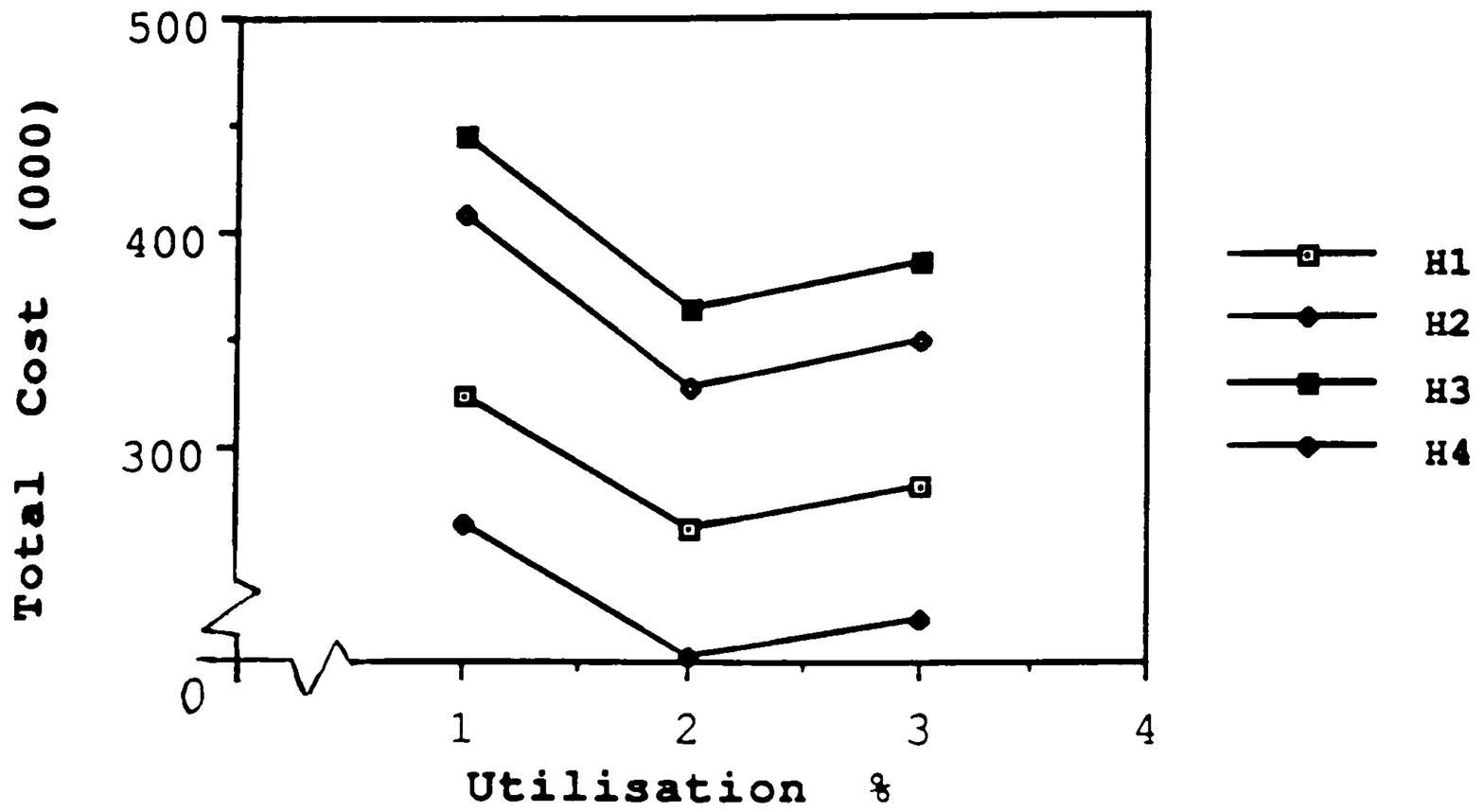
Graph 5.8.1. One End Item Problem with Structure 1



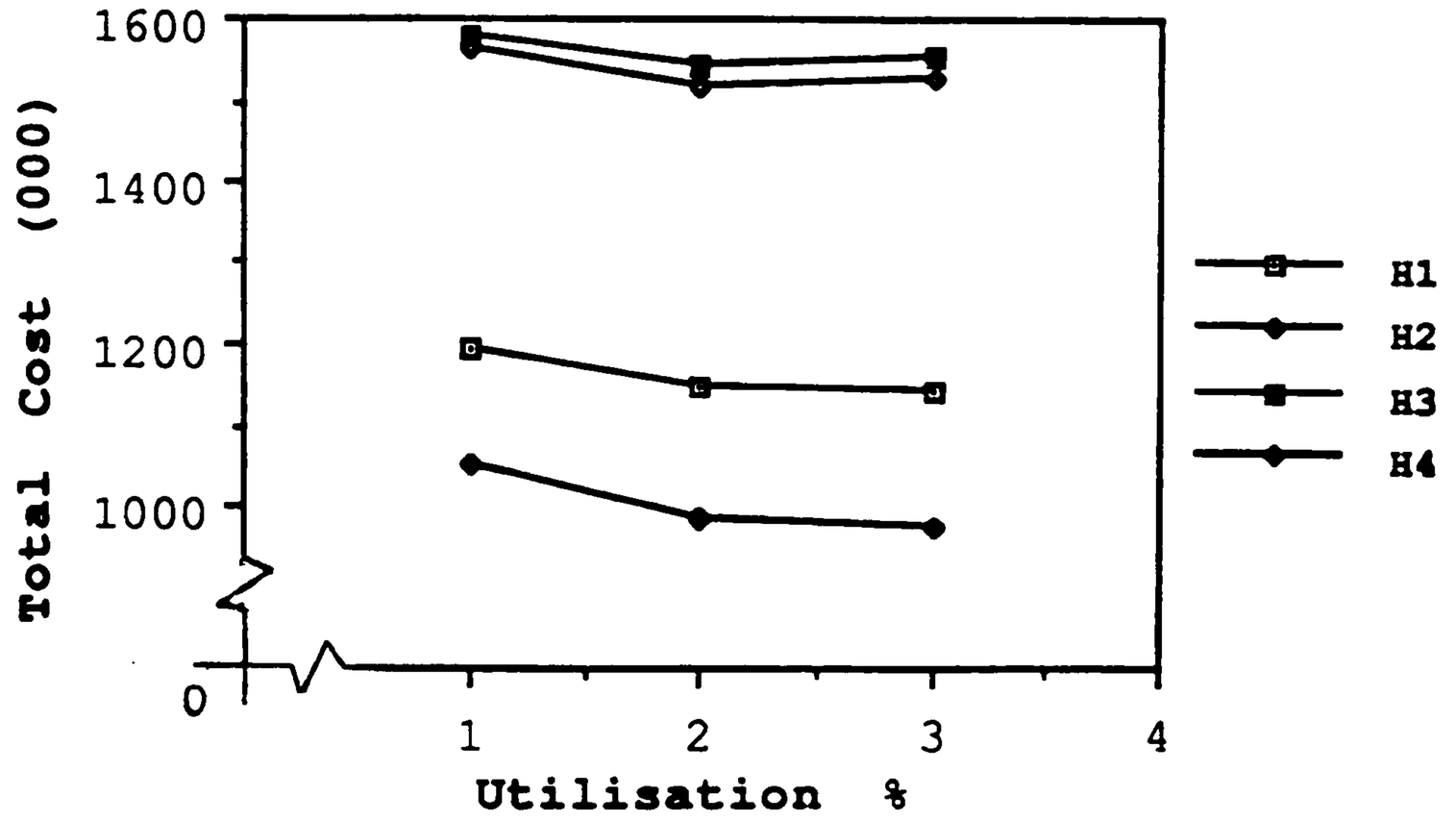
Graph 5.8.2. One End Item Problem with Structure 2



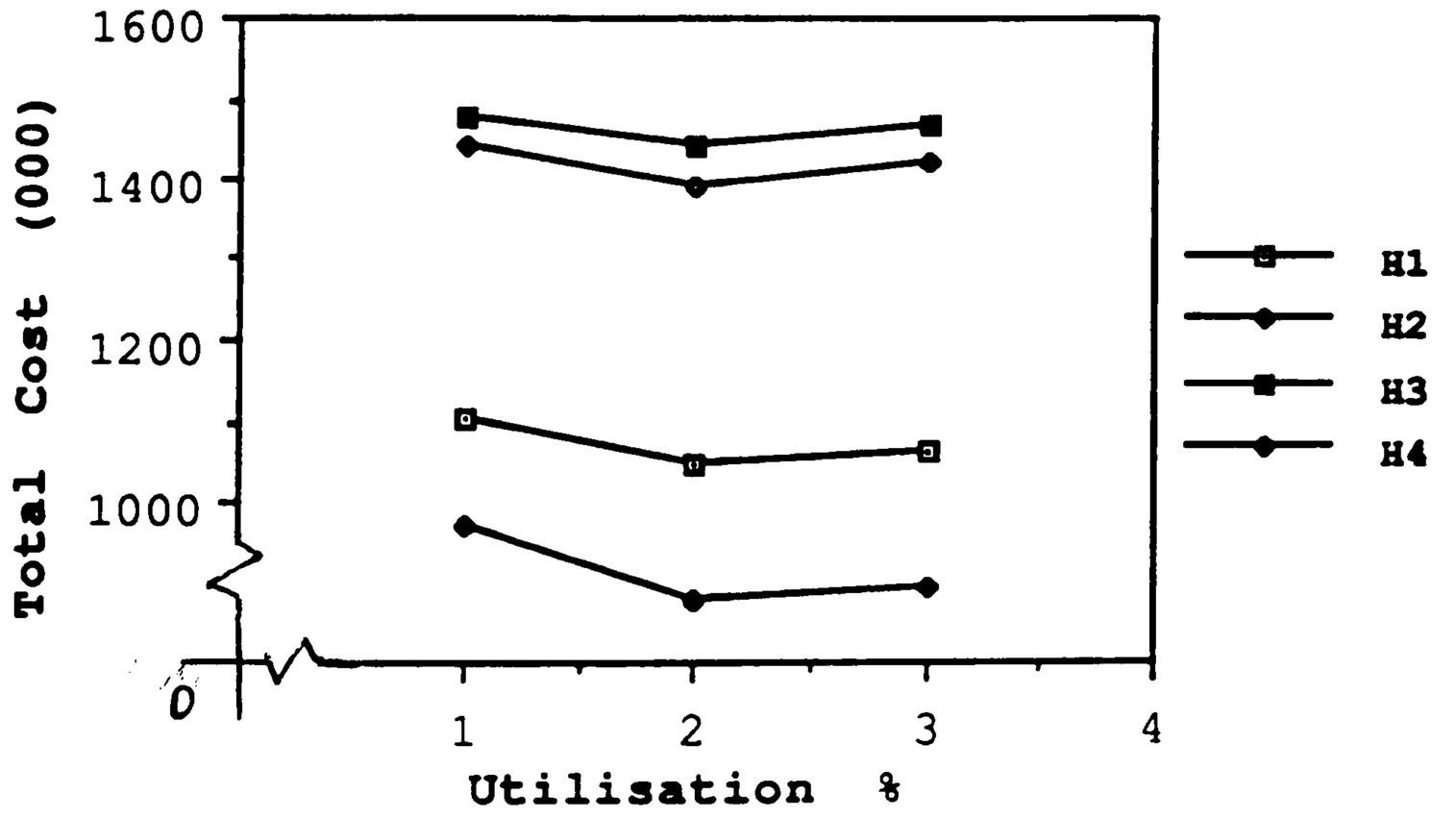
Graph 5.8.3. One End Item Problem with Structure 3



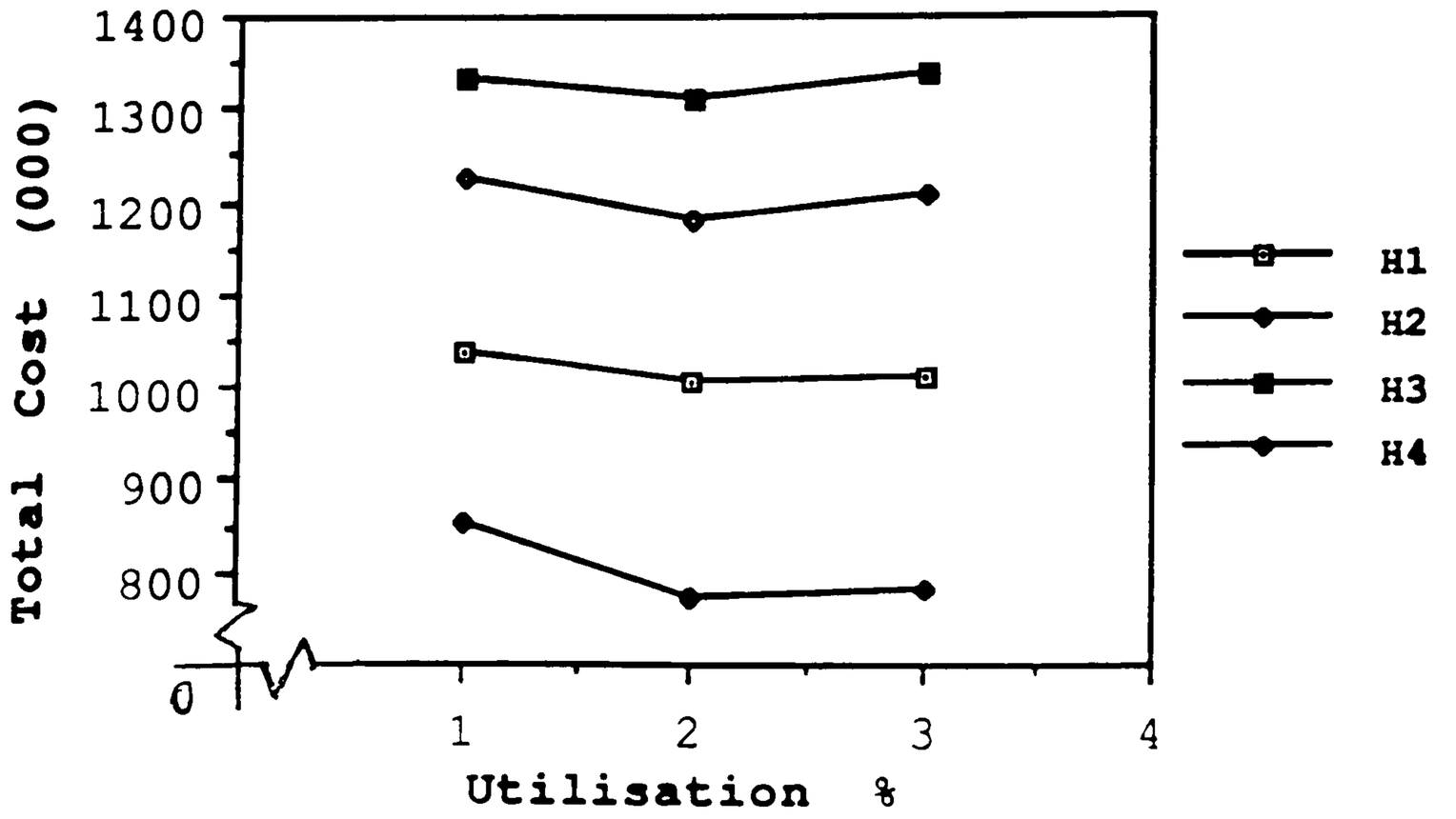
Graph 5.8.7. Five End Item Problem with Structure 1



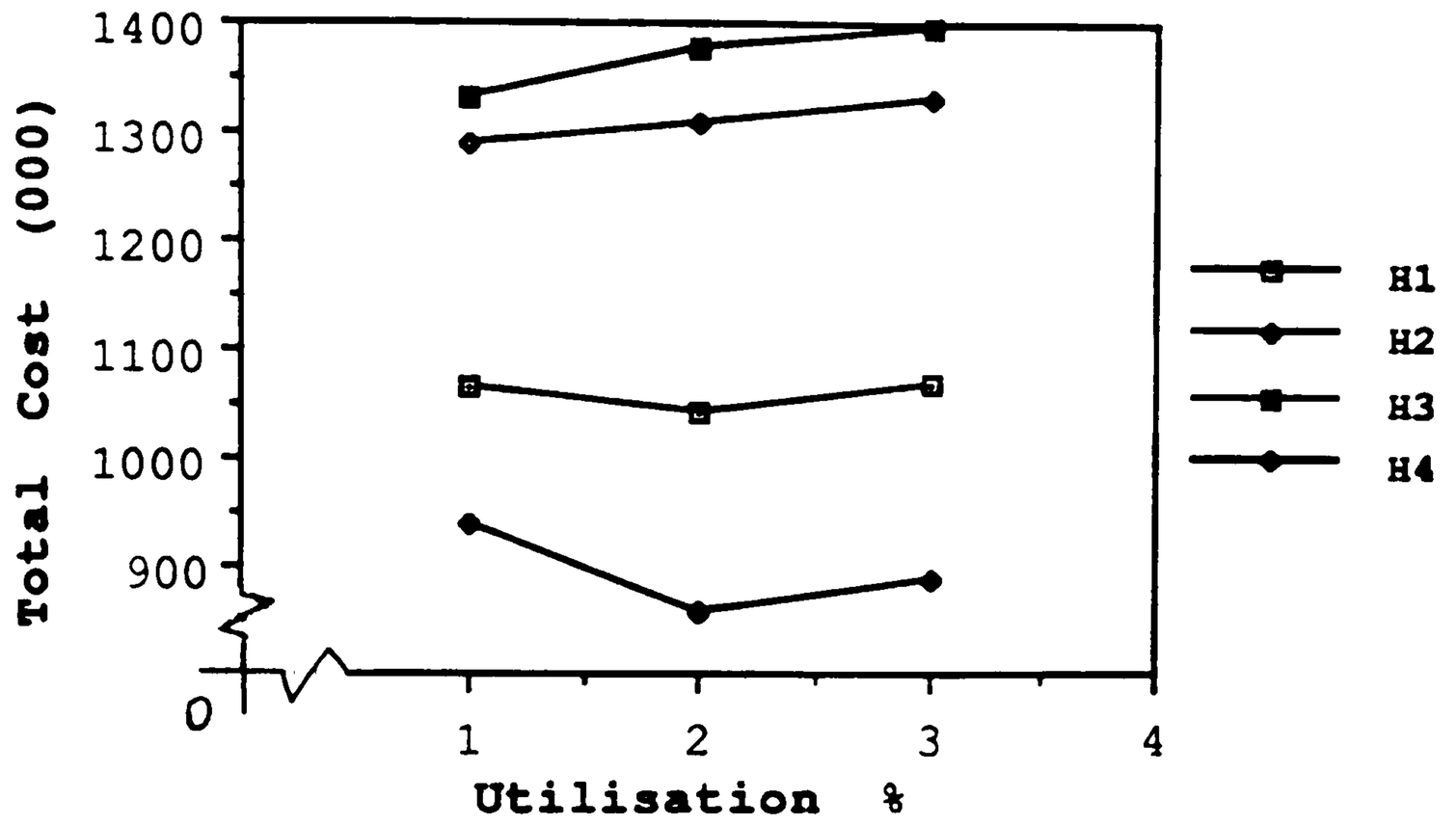
Graph 5.8.8. Five End Item Problem with Structure 2



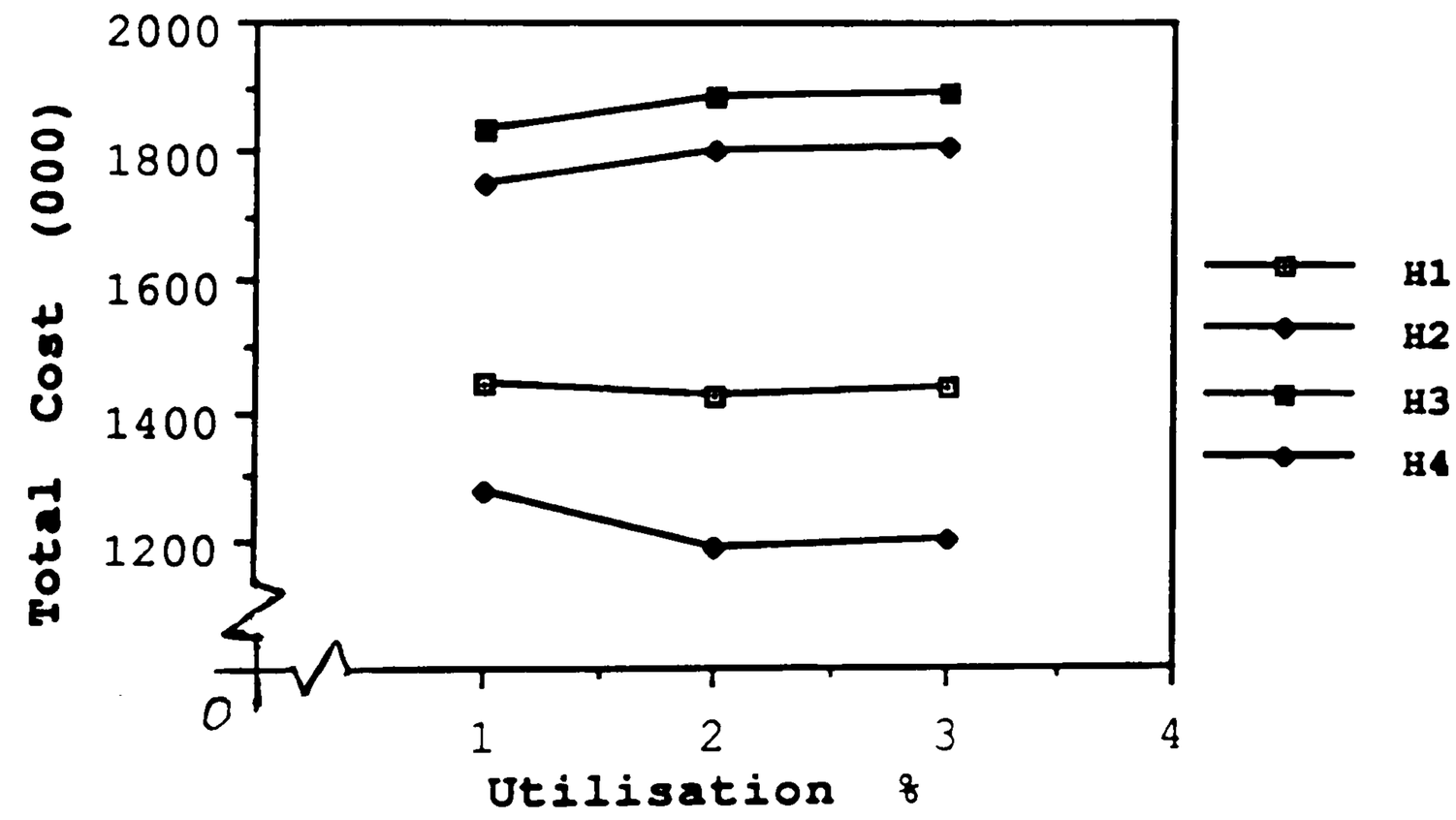
Graph 5.8.9. Five End Item Problem with Structure 3



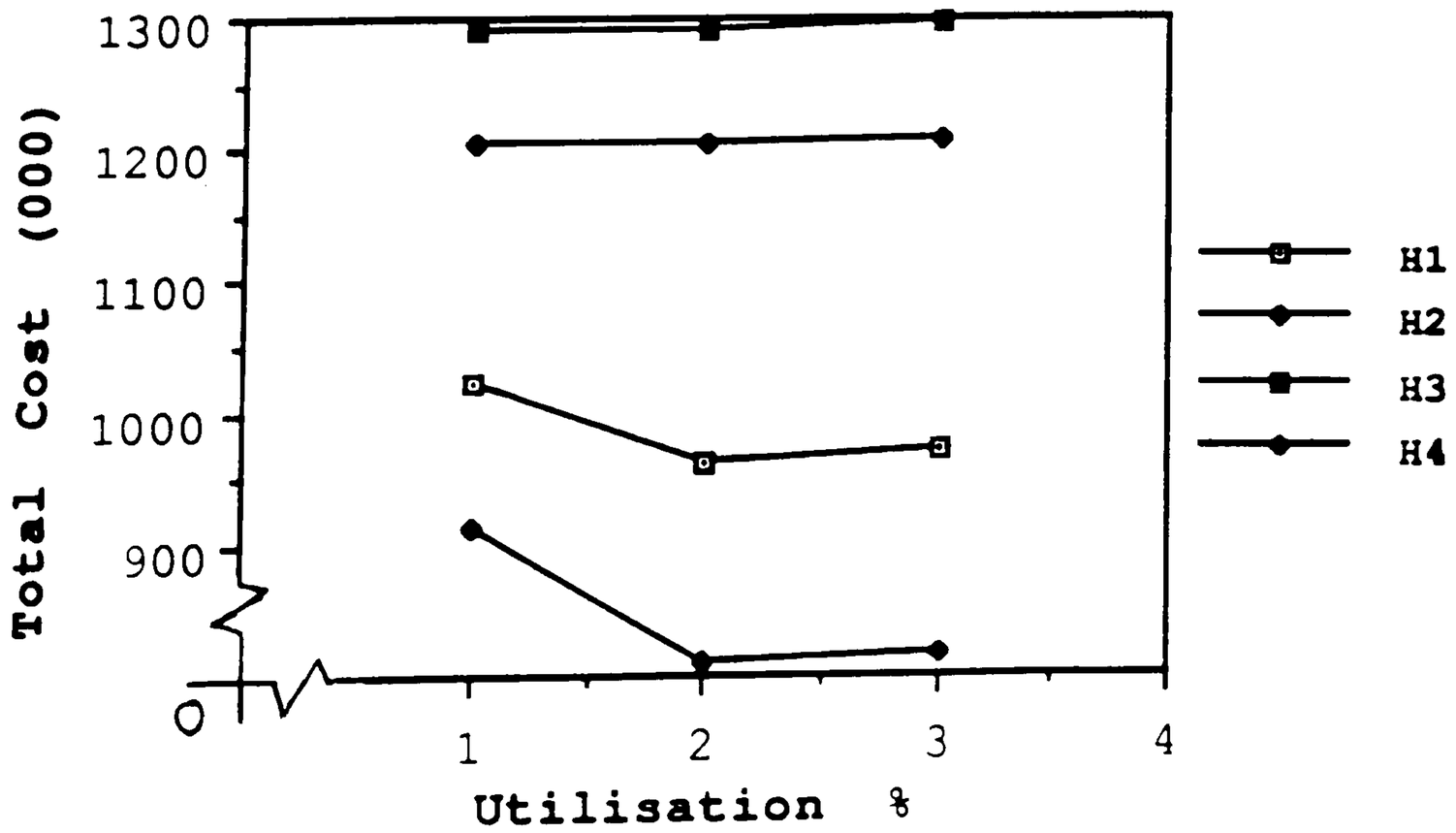
Graph 5.8.4.Three End Item Problem with Structure 1



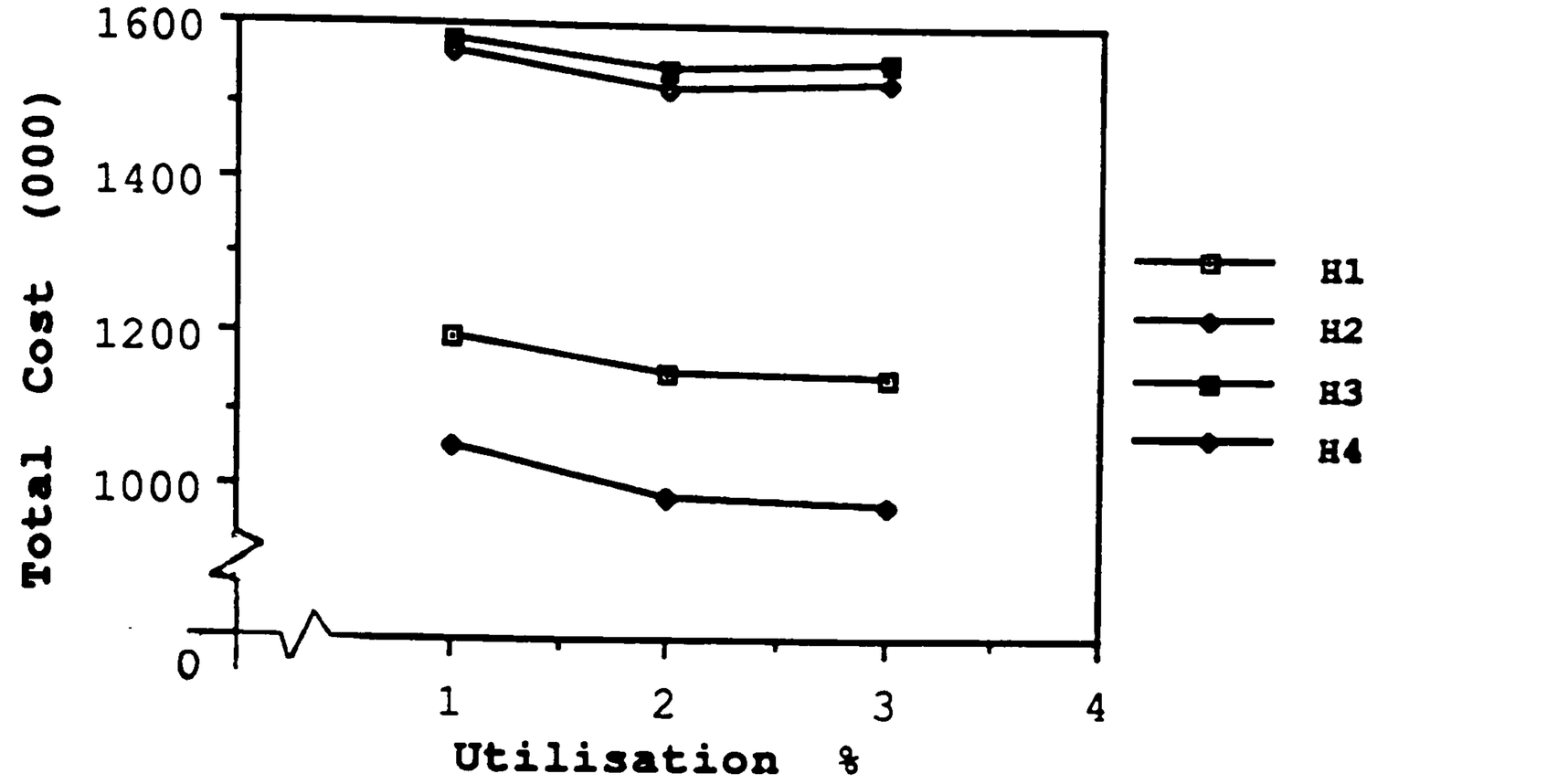
Graph 5.8.5.Three End Item Problem with Structure 2



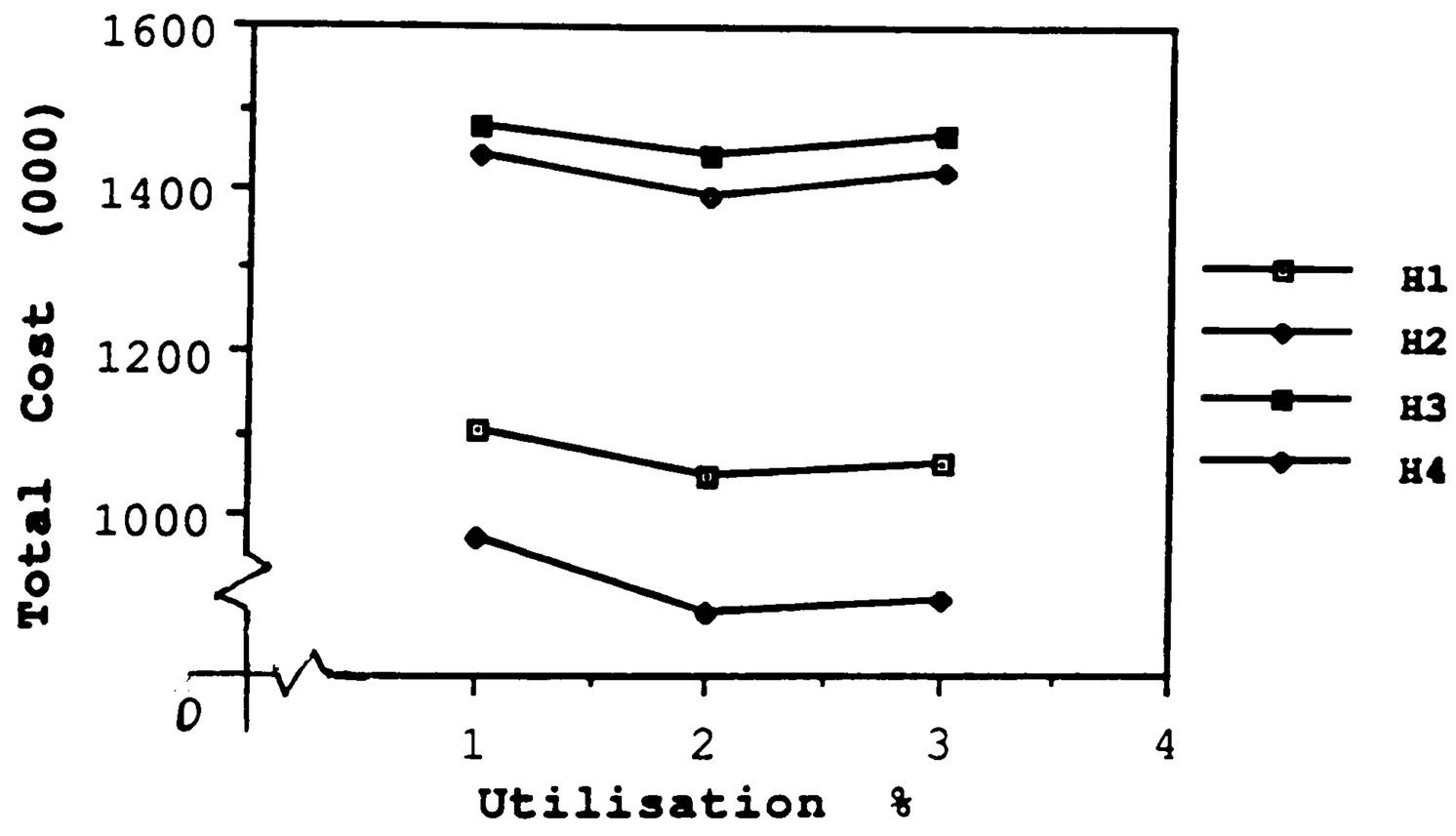
Graph 5.8.6.Three End Item Problem with Structure 3



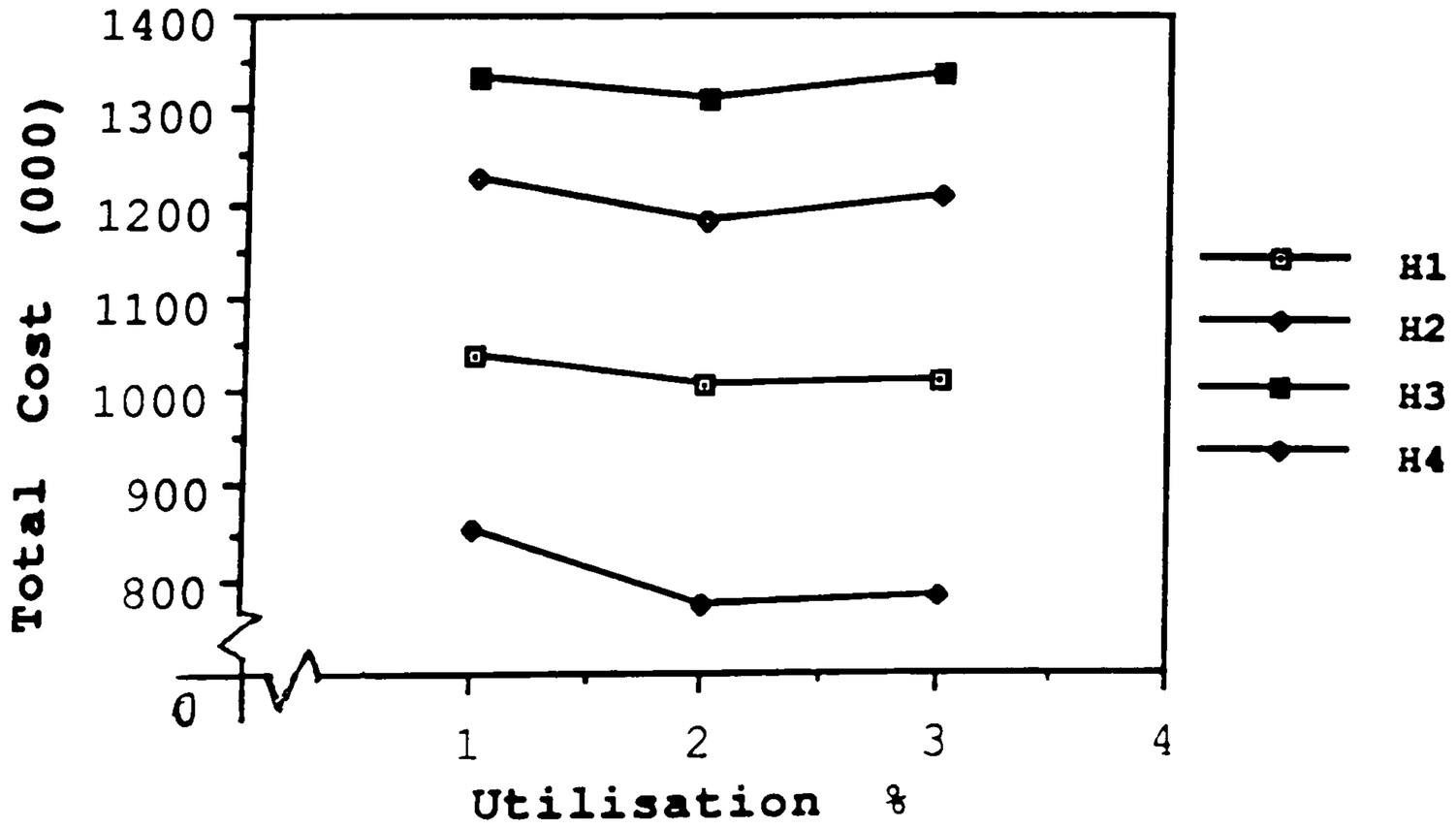
Graph 5.8.7. Five End Item Problem with Structure 1



Graph 5.8.8. Five End Item Problem with Structure 2



Graph 5.8.9. Five End Item Problem with Structure 3



CHAPTER 6

RESULTS AND DISCUSSIONS

6.1. Introduction

Chapter 4 proposed a simple heuristic to solve multi-level lot-sizing problem when there is a bottleneck. It also illustrated that if the product items are categorized into:

(1) end-items, constrained by the bottleneck
(2) non-end-items, unconstrained

then simple procedures can be used independently to determine the production quantities required. A heuristic was developed to solve the constrained product items. The EOQ and Silver-Meal approaches were employed on the unconstrained product items. The above procedures were also applied to the multiple bottleneck case in chapter 4 and to the rolling schedule problem in chapter 5. The procedures assumed that all the product items would be treated independently so that they could not affect each other's production. These procedures may be applied to the dependent product item cases but that is a project for further research. The results and discussions of multi-level lot-sizing problem with bottleneck(s) will be given first (section 6.2) followed by the results of the rolling schedule problem (section 6.3). The worst case analysis of the heuristic will be given in section 6.4. Finally the conclusion will be provided in section 6.5.

6.2. The Results and Discussions of Multi-Level Lot-Sizing Problem with Bottleneck(s)

The integer programming formulation solved by using the software MGG [1987] and SCICONIC [1986], and the heuristic approach of chapter 4, coded in Fortran 77, were compared on sets of data obtained from Billington [1983] and also discussed in Billington et al. [1986]. In the experimental studies three different cost structures, three demand streams and three capacity levels were used. The three cost structures are detailed in Billington [1983]. Essentially for each structure the holding costs and the setup costs are set to different levels such that for structure 1 the average of the holding costs is 0.92 and the average for the setup costs is 360. For structures 2 and 3 the corresponding values are 1.31, 360 and 0.79, 380 for holding cost and setup cost, respectively. The three demand streams are generated to give a low, medium and high coefficient of variation, C_v, which is defined as the demand standard deviation divided by mean demand. That is

$$C_v = \frac{\sqrt{\sum_{i=1}^{n} \frac{(d_i - \bar{d}_i)^2}{n-1}}}{\sum_{i=1}^{n} \frac{d_i}{n}}$$

In another words the coefficient of variation or C_v is the demand standard deviation expressed as a proportion of mean demand. A low coefficient of variation, C_v = 0.1144, a medium coefficient of variation, C_v = 0.1926, and a high coefficient of variation C_v = 0.3955 for the three-end-item problem were used in Billington et al. [1986]. From now on the levels of the coefficient of variation will be referred to as structure 1, structure

2, and structure 3. The coefficients of variation for one- and five-end-item problems can be seen in Appendix B. Capacity is set after demands are determined such that the total demand is divided by the product of the percentage of utilisation (low (50%), medium (75%), high (95%)) and the number of time periods.

The results of the experiments for the single bottleneck problem, with the original costs and a reduction of the original costs by 20% and 40% in turn, are illustrated in Tables 4.6.1-4.6.9 (81 problems were investigated in all). The same results in those tables are also presented in Graphs 4.6.1-4.6.9. In addition, the results of 20% reduction in each of the original holding and setup cost are depicted in Tables 4.6.10-4.6.12 (27 problems were investigated). Different capacity utilisations were investigated, such as 60% and 80%, to show how the heuristic worked for those cases and the results of 12 problems are depicted in Tables 4.6.13-4.6.18.

The results of the multiple bottleneck cases are shown in Tables 4.6.19-4.6.21. The same results are also illustrated in Graphs 4.6.10-4.6.12.

In each table, the details of the linear programming (LP) solution relative to the integer programming (IP) formulation are given in the third column. Details of the integer programming branch and bound approach are given in the fourth column. Details of the heuristic solution for the constrained product items with the EOQ and Silver-Meal for the unconstrained product items are given in the fifth and sixth columns respectively for different capacity utilisation.

In each graph (Graphs 4.6.1-4.6.9) the linear programming solution (LP), the integer programming solution (IP), and the heuristic solution for the constrained product items

with either Economic Order Quantity (H1), or the Silver-Meal approaches (H2) for the unconstrained product items are depicted in the legends. The current values, the CPU time and the level of optimality are illustrated on the vertical axis while the capacity utilisations (set at 50%, 75%, or 95%) are depicted on the horizontal axis. In addition in Graphs 4.6.10-4.6.12, the numbers on the horizontal axis refer to multiple bottleneck cases where the bottlenecks occurred in the final item(s) and intermediate item(s). The utilisations which applied to the multiple bottleneck cases were 50% and 25%, 75% and 50%, 95% and 75% of capacity utilisation at the final product item(s) and intermediate product item(s) respectively.

The branching process by which the integer programming solutions were obtained was the standard default of the SCICONIC software which comprises an approach to choose sub-problems which minimise the percentage error in the degradation of the objective function. The "dynamic presolve" option of SCICONIC was also used which aids branching exploration by tightening bounds where possible. The IP and heuristic solutions are compared to the LP optimum to give some indication of the quality of the solutions as the LP optimum provides a lower bound to the solution to the problem. In all cases the IP solution is not a proven optimal solution and the branch and bound process had to be cut off, before optimality could be proved. The cut off occurred once a large amount of computer time had elapsed and further effort appeared unproductive. Computation was stopped after approximately 3000 branch and bound nodes had been explored. The reason for this was that although a number of problems were run over 10000 or more branch and bound nodes, it was found that no better solution was obtained than in the first 2000-3000 nodes, so 3000 nodes was taken as a convenient stopping point. CPU times quoted are for the time (in

seconds) taken on a Hewlett Packard 9000 to reach the given solution. A number of features are evident from the results quoted in the tables.

1. The heuristic approach is rapid, taking a few seconds of CPU time.

2. In all but a few cases (such as 95% utilisation for structure 2 in one-end-item problems, and structure 2 for three-end-item problems) the solutions using the heuristic for the constrained product items with Economic Order Quantity for the unconstrained product items are better than the IP solutions. The solutions using the heuristic for the constrained product items with Silver-Meal for the unconstrained product items are better than the IP solutions for all cases except for structure 2 for the three-end-item problems with multiple bottlenecks. Also using the heuristic for constrained product items with the Silver-Meal approach for unconstrained product items provides a better solution than using the heuristic for constrained product items with the Economic Order Quantity approach for the unconstrained product items except for structure 1 for the one-end-item problem. The reason is that the Silver-Meal approach yields better results with varying demand. (see for example Monks [1987]).

3. The heuristic solution for the constrained product items with the EOQ and Silver-Meal approaches for the unconstrained product items yields the same optimality when the original setup and holding costs were reduced by 20% and 40% in turn (see Tables 4.6.1-4.6.9), but it does not give the same reduction when each of the original holding and setup costs were reduced by 20% (see Tables 4.6.10-4.6.12).

For the five-end-item problems the heuristic seems to work particularly well, in the sense that the solutions are very close to the lower bound value and sometimes the same as the lower bound for structure 2 and structure 3. However, this may only mean that the lower bounds are tighter for these problems. These tighter lower bounds may arise because more realistic values of the effect of setup costs (which are unrealistically reduced in the LP relaxation of the IP formulation) are likely to arise when there are more types of product items being produced in the bottleneck and so there are more fractions of the true setup cost to add together.

For the few cases in three-end-item problems (structure 2) and one-end-item problems (95% utilisation for structure 2) where the heuristic does not work well it may be that rather more setups than necesary are being used when the setup cost is above average.

The IP solutions obtained were not optimal, but it should be noted that the work of Billington et al. [1986] was also unable to make comparison of its Lagrangean solutions with optimal solutions.

The LP solution (LP) is a lower bound on total cost. The optimal IP solution (IP^*), if known, would be larger than the LP solution because the LP solution involves fractions.

The heuristic solution is a solution which is not guaranteed to be optimal, but is integer feasible.

Ordinarily it would be expected to discover the hierarchy

$$LP \leq IP^* \leq \text{Heuristic solution}$$

However, in the use of the problems tested in this thesis, the optimal IP solution is not known and the best IP solution obtained by branch and bound method may be substantially poorer than the optimal because production scheduling problems are so hard for the branch and bound method to solve. Thus in the results described it was generally found the hierarchy was

$$LP \leq \text{Heuristic Solution} \leq IP$$

where IP is the best (non-optimal) solution found by branch and bound method. Thus, in fact, the heuristic generally gives a better integer solution than the IP solution and if it could have been easily inserted into the SCICONIC code it could have improved the branch and bound search. All solution approaches solve the same formulation of the problem (given in chapters 3 and 4).

It is impossible to compare precisely the heuristic approach of chapter 4 with that of Billington et al. [1986] as we do not have access to their program. It was felt that once rapid solution times for the heuristic approach were obtained (all less than 3.31 CPU seconds) and such high quality solutions were obtained for the five-end-item problems that it was not appropriate to program the Lagrangean approach. What was of interest was to find a simpler approach. It is likely that their method produces good quality solutions but the code is complex and unlikely to operate as rapidly as the few seconds required by the heuristic in this thesis. The approach is sufficiently flexible to provide quick solutions for a variety of extensions of the basic problem and the approach is appropriate for quick reworking of schedules whenever changes occur in demand streams or breakdowns occur.

6.3. The Results and Discussions of Rolling Schedule for Multi-Level Lot-Sizing Problem with Bottleneck(s)

The heuristic for the constrained product item(s) with the EOQ and Silver-Meal approaches for the unconstrained product items for the rolling schedule was coded in Fortran 77, and three demand streams were generated using NAG Subroutines [1990] for the Normal Distribution on a Hewlett Packard. As in the previous section, three different cost structures, three demand streams and three capacity levels were employed. The average holding and setup costs were set to different levels as follows: For structure 1, the average holding cost is 0.92 and the average setup cost is 360. For structure 2, the average holding cost is 1.31 and the average setup cost is 360. For structure 3, the corresponding values are 0.79 and 380 for the average holding and setup costs.

Three demand streams for the three-end-item problems were generated to give a low (C_v = 0.1144), a medium (C_v = 0.1926), and a high (C_v = 0.3955) coefficient of variation for each of ninety-six period demand data. The C_v is the standard deviation as a proportion of mean demand. Capacity is set after demands are determined such that the total demand is divided by the product of the percentage of utilisation and the number of time periods. The capacity utilisations varied from 25% to 95% in the experimental studies.

The results for the rolling schedule are shown in Tables 5.8.1-5.8.9 with different capacity utilisation (27 problems were investigated). In each table, the total cost of the ninety-six period problem using the heuristic for the constrained item(s) with the EOQ and Silver-Meal approaches for the unconstrained items are shown. The second and third columns refer to problems without a

rolling schedule and the fourth and fifth columns refer to problems with a rolling schedule.

Graphs 5.8.1-5.8.9 are presented in the same way as the results above. In each graph, the number 1 on the horizontal axis refers to the multiple bottleneck cases. It is assumed that those bottlenecks occurred in the final product item(s) with 50% capacity utilisation and the intermediate product item(s) with 50% capacity utilisation. Further, the numbers 2 and 3 on the horizontal axis refer to single bottleneck cases which occurred at the final product item(s) with 50% and 75% utilisation, respectively. In addition, the legends H1, H2 refer to the total cost without the rolling schedule, and H3, H4 refer to the total cost with the rolling schedule.

The fixed planning period is used to solve different problems in chapter 4, ignoring the more realistic conditions dealt with by dynamic studies in chapter 5.

In the experimental studies, such as in Blackburn et al.[1982], four to ninety-six forecast windows and a six period rolling schedule horizon were used. The heuristic for the constrained product items and the EOQ and Silver-Meal approaches for the unconstrained product items were used to solve the problems using the rolling schedule. The rolling schedules were formed as follows: First, the problem for 1 to N is solved, but only the first decision, for the periods 1 to k, was implemented. The process was repeated for periods k+3, k+6,..., k+N, continuing until the production decision was made in all of the ninety-six periods. Although the changes after every 3 periods are examined in this thesis, the program which is reported in Appendix C can provide the changes after each time period.

The results in Tables 5.8.1-5.8.9 illustrate that increasing the planning horizon results in extra costs, however more information is generated in the planning period which can make stable plans. The results also show that the Silver-Meal approach performed better than the EOQ. The reason is because the Silver-Meal approach allows for more stocks at the beginning of the new periods which are subsequently beneficial.

6.4. Worst Case Analysis of Heuristics

In the literature it is very difficult to solve the capacitated lot-sizing problems with exact techniques such as linear programming or integer programming, because these problems take substantial computer time. These problems are known as NP-hard problems as was mentioned briefly in chapter 4. They are hard problems because optimal techniques are unable to solve the problems in a reasonable amount of computer time. Because of these difficulties attention turned to heuristics which find approximate solutions for those problems. Heuristics can solve difficult problems satisfactorily, see for example Fisher [1980], Garey and Johnson [1979], Iyogun [1991]. In this thesis the computation time took a maximum of 3.31 seconds, on the other hand integer programming solutions took a maximum of 887.08 seconds to solve the same problem. When heuristics are applied to difficult problems, we have to answer the question "How good are the heuristics?" The worst case analysis can help to answer this question. The worst case analysis is defined as a maximum deviation from the optimality which can happen if a heuristic is applied to the problem sets. The aim of this analysis is to predict the heuristic performance so that the heuristic performance should be viewed as complementary instead of competitive as explained in Fisher [1980]. According to the definition, the heuristic performance can be found using the

following equation which is again to be found in Fisher [1980]:

$$Z_H\ (I) \leq r\ Z\ (I)$$

where Z_H (I) is the heuristic solution for the problem I, Z (I) is the optimal solution for the problem I, and r is the heuristic performance. The best heuristic is the smallest ratio for a given problem . That is,

$$Z_H\ (I) = r\ Z\ (I)$$

This heuristic performance refers to optimality in the experimental cases in Tables 4.6.1-4.6.21. In that case the best heuristic performance is involved in structure 2 and 3 for the five-end-item problem (r = optimality = 1 in Tables 4.6.8, 4.6.9, 4.6.12, 4.6.18), and the worst case performance among all problems was the structure 2 for the three-end-item problem in the experiment (r= optimality = 1.360 in Table 4.6.20). The reason for the poor performance of the heuristic was explained in section 6.2.

In this section, the worst case analysis is explained in terms of setup cost and holding cost. The heuristic, developed in chapter 4, works well when the setup cost is high and the holding cost is low, so the question arises "what is the worst case for this data?". To answer this question two opposing heuristics will be introduced briefly:

1. The heuristic, developed in chapter 4, which aimed to produce as much as possible for each product item for a period or several periods within a system with a bottleneck.

2. The opposite to heuristic 1, which is to, produce as little as required within each period for each product item.

Let TC1 and TC2 denote total cost in relation to the heuristics above. As will be seen from the second approach there is no holding cost in these cases. The tradeoff point between setup and holding costs would be as follows:

$$TC1 = TC2$$

$$S_i\ \delta_1\ (P_{it}) + h_i\ I_{it} = S_i\ \delta_2\ (P_{it})$$

$$\text{then } h_i = (S_i\ \delta_2\ (P_{it}) - S_i\ \delta_1\ (P_{it}))\ /\ I_{it}$$

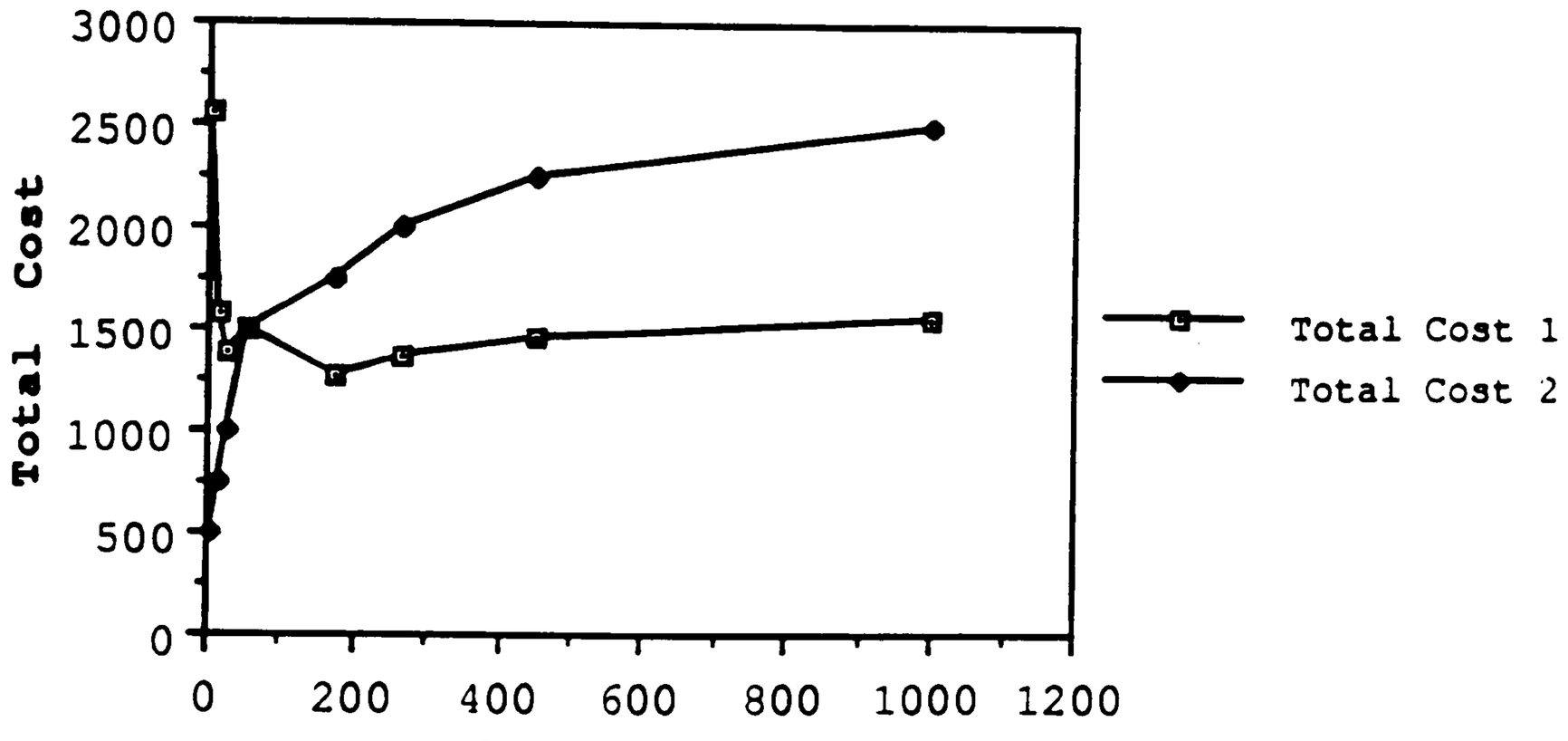
where $\delta_1\ (P_{it})$, and $\delta_2\ (P_{it})$ are the production indicators according to the heuristics, h_i, S_i, I_{it} are the holding cost, setup cost and inventory level for product item i during period t respectively. A small example is given below:

Table 6.4. A Cost Analysis

Setup Cost	Holding Cost	The Ratio of Costs (Setup / Holding)	Total Cost 1	Total Cost 2
500.0	0.5	1000.0	1556.5	2500.0
450.0	1.0	450.0	1463.0	2250.0
400.0	1.5	266.7	1369.5	2000.0
350.0	2.0	175.0	1276.0	1750.0
300.0	5.3	56.5	1500.0	1500.0
200.0	7.0	28.6	1391.0	1000.0
150.0	10.0	15.0	1580.0	750.0
100.0	20.0	5.0	2560.0	500.0

The tradeoff point between the setup cost and the holding cost is $h_i = 0.017\ S_i$ according to the example above. It means that the heuristic developed in chapter 4 works

well when the setup cost is greater than or equal to 56.5 times the holding cost in this case. The relations between the setup cost and holding costs can be found using the above formulation. The relation between these two costs is ilustrated in Graph 6.4.



Graph 6.4. The Relation Between Setup and Holding Costs.

6.5. Summary and Conclusion

In this chapter first the results of single and multiple bottleneck problem cases were discussed. The results showed that the heuristic worked well for the five-end-item problem. This was because the solutions were so close to the lower bound value (LP). It meant that the lower bounds were tighter for those problems. The reason why the heuristic did not work well for a few cases, such as structure 2 with 95% capacity utilisation in one-end-item problem and structure 2 in three-end-item problem, may be that rather more setups than necessary were being used when setup cost was above average.

Then, the results of chapter 5 was discussed. The 96 period problem were investigated to show the effect of the rolling schedule environment. The results showed that when the uncertainties increased, total cost increased. The results also illustrated that the Silver-Meal approach is better than the EOQ because of varying demands.

Finally, the worst case analysis considered how good the heuristic was. The analysis was done in respect of setup cost and holding cost. The idea was to find the relations between those two costs to minimise the total cost. For that reason a very simple example was given and the result showed that the heuristic performed well when the setup cost was greater than or equal to 56.5 times of holding cost according to the data above. This was the tradeoff point between those costs for that example problem. On the other hand when this proportion increased, the solution became worse.

In conclusion, the heuristic provides a quick and easy solution for the problems and is sufficiently simple to be used even without a computer routine.

CHAPTER 7

A COMPARISON FOR THE ASSEMBLY SYSTEM

7.1. Introduction

The results of chapters 4 and 5, and also the worst case analysis were discussed in the previous chapter. This chapter first shows, as was mentioned in chapter 2, the application of the heuristic of Eftekharzadeh [1988] to the assembly system in order to compare it to the heuristic of this thesis (section 7.2). Section 7.3 then provides a sensitivity analysis while comparing the setup and holding cost. Finally some conclusions are drawn in section 7.4.

7.2. A Comparison for Assembly Systems

In this section, the heuristic of Eftekharzadeh [1988] is compared to the heuristic of this thesis for the assembly product structure, where each part may have only one successor but many predecessors. This heuristic was chosen for particular consideration because it represented a new development that related well to the work of this thesis. The product structure is illustrated in Figure 7.1.

Eftekharzadeh [1988] in fact proposed two heuristics: which are: (1) Selective Enumeration; and (2) Modified Per Period heuristics for multi-stage lot-sizing assembly systems. The Modified Per Period heuristic, which is an extension of the multi-stage uncapacitated procedure, can

achieve the feasible schedule by shifting the production from the period with insufficient capacity to the immediate left. In this chapter, the Selective Enumeration procedure will be explained in detail because the results of the two heuristics by Eftekharzadeh are essentially the same. The Selective Enumeration heuristic is the conversion of the uncapacitated multi-stage lot-sizing heuristic of Afentakis [1987] to a capacitated lot-sizing heuristic. This heuristic starts to solve the problem by using the shortest path procedure for the t period problem by augmenting the solution to the (T-1) period problem according to the forward fashion and selects the plan of minimum cost for each period and puts it in ascending order. That is, the one period problem is solved first, then the two period problem, then the three period problem and so on for each stage. In conjunction with the heuristic there are 2^{t-1} schedules at each period t for each stage and these schedules should satisfy two conditions. Firstly the capacity condition, which requires that available capacity must not be less than the requirements, and secondly the system condition which requires that the cumulative production at the end of period should be at least equal to cumulative production for its successor stage A(i) for any two adjacent stages. The schedule which provides the minimum cost is selected so that the selected plan remains feasible for the whole system. On the other hand some of the plan at any period t might be infeasible. Such plans that have the infeasible parts must be deleted from the network. This heuristic when compared to our heuristic is quite similiar to the Afentakis [1987], and Karni and Roll [1982] except that Karni and Roll [1982] starts the planning with the Wagner-Whitin solution for each item, and Afentakis [1987] uses uncapacitated lot-sizing procedure.

Some changes to the Eftekharzadeh approach were found to be necessary to handle certain situations which were apparently overlooked in the steps of the heuristic (these include simplifications also).

The steps of the above heuristic are as follows:

Step 1. Create a network for each stage i (i = 1,...,N) with all the possible q period problems (q = 1,...,t). Calculate the costs for each q period problem using the shortest path procedure and put them in ascending order for each t-period problem (t ≤ T-1). These costs are calculated by the following formula:

$$\text{Min} \sum_{i=1}^{N} \sum_{t=1}^{T} \{ cs_i * \delta(P_{it}) + h_i * I_{it} \}$$

The notation is the same as before.

Step 2. Check the feasiblity for each period at any stage with reference to the capacity and system conditions starting with the first period. The capacity condition is:

$$\sum_{i=1}^{N} [b_i * P_{it}] \leq Cap_t$$

In each period, the possible schedules which provide the minimum cost are combined to create the final schedules. If there is any infeasible schedule at any period, then that plan(s) is deleted from the network, and the next lowest cost plan replaces the deleted schedule. This will be explained in detail later in the chapter. The b_i value is the time needed to produce one unit of product i in the above formulation and this is set to 1.

Step 3. Generate the $N(i) = t + 2 - t_i$ schedules for the T period problem, where N(i) is the number of stages and t_i is the last production period in the plan V(i, t). The alternative plans, k, are constructed by adopting the plan computed by the heuristic from period 1 through $t_i + k - 2$. Then the k^{th} schedule W(i, k) is:

$$W(i, k) = \{V(i, t_i + k - 2), \sum_{r=t_i+k-1} R_{ir}, 0, 0\} \quad \begin{matrix} k=1,..,N(i) \\ i=1,..,N \end{matrix}$$

and the related cost is calculated by using the above cost function.

Step 4. Consider all possible schedules which yield the minimum cost, and check whether the schedules satisfy the system and capacity conditions. If it is true, then stop. If it is not, then go to step 3.

The data used for this method is depicted in Table 7.1 and Table 7.2 respectively, and also the assembly product structure in Figure 7.1 (adopted from Eftekharzadeh [1988])

Table 7.1. Data for a 5-period problem.

Stage	Successor	Predecessor	Setup Cost	Echelon Holding Cost
i	A(i)	B(i)	S_i	e_i
1	-	2,3	10	1.0
2	1	-	20	1.0
3	1	-	8	1.0

The echelon holding cost in Table 7.1., explained in chapter 2, is the cost charged to each unit of the echelon inventory which is all units in the system at stage j and its successors. It is calculated according to the following formula

$$e_i = h_i - \sum_{j \varepsilon C(i)} h_j$$

where C(i) is the set of immediate predecessors of product item i.

Table 7.2. Demand and capacity for a 5-period problem.

Period	Demand	Period Capacity
1	8	39
2	15	39
3	4	39
4	8	39
5	12	39

According to Table 7.1, the product structure becomes:

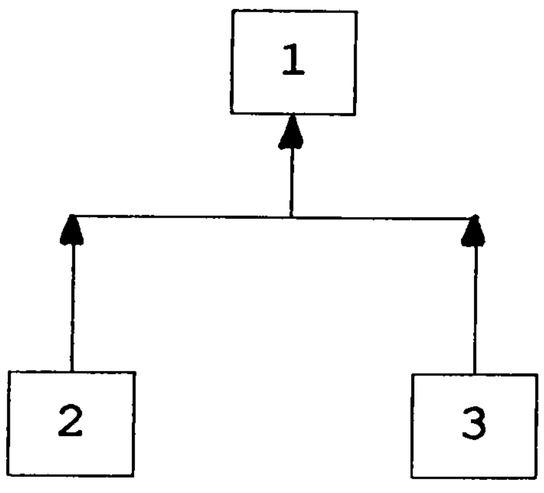


Figure 7.1. The three stage assembly systems.

Eftekharzadeh [1988] started to solve the problem for the q (q = 1,..., t) period problem for stages 1, 2 and 3 to determine the production schedules. The schedules are created according to the forward fashion, that is, first the one period problem is solved, then the two period problem, and so on. The schedules and costs for the first four periods are illustrated in Table 7.3. The costs are the sums of the setup costs and echelon holding costs. For example the cost of 24 in period 3 for stage i consists of two setup costs (20) plus the echelon holding costs for four units in period 2.

Table 7.3. Schedules and costs for the 4-period problem.

Stage	Period			
i	1	2	3	4
1	(8)[10]	(8,15)[20]	(8,19,0)[24]	(8,19,0, 8)[34]
2	(8)[20]	(23, 0)[35]	(27, 0,0)[43]	(8,27,0, 0)[60]
3	(8)[8]	(8,15)[16]	(8,19,0)[20]	(8,19,0, 8)[28]
	(24)[38]	(39,30)[71]	(43,38,0)[87]	(24,65,0,16)[122]

For step 2 the feasibility is checked (using the formula in step 2). As will be seen from Table 7.3 there are two infeasible schedules which come from the combined schedules for the 3 and 4 period problems. They are 43 in period 1 for the 3 period problem, and 65 in period 2 for the 4 period problem, therefore the schedule for which the maximum is produced in that particular period for product item i must be deleted from the network; e.g. product item 2 in the above Table 7.3, which is 27 in period 1 for the 3 period problem, and the schedule that gives the next minimum cost will be appended to the network. If there is the same maximum quantity for different product items, then the schedule which will be deleted from the network is arbitrary. The new schedule is illustrated in Table 7.4.

Table 7.4. Schedules and costs for the 4-period problem.

Stage	Period			
i	1	2	3	4
1	(8)[10]	(8,15)[20]	(8,15,4)[30]	(8,15, 4, 8)[40]
2	(8)[20]	(23, 0)[35]	(23, 0,4)[55]	(23, 0,12, 0)[63]
3	(8)[8]	(8,15)[16]	(8,19,0)[20]	(8,19, 0, 8)[28]
	(24)[38]	(39,30)[71]	(39,34,8)[105]	(39,34,16,16)[131]

The feasible schedule is found in step 2, hence the schedule for the next period will be generated. For this reason, the alternative plan is calculated for each stage in step 3 in the following way.

$N(i) = t + 2 - t_i$ $\quad i = 1,\ldots,3$

$N(1) = 4 + 2 - 4 = 2$, $N(2) = 4 + 2 - 3 = 3$ and

$N(3) = 4 + 2 - 4 = 2$

the alternative plan is as follows;

$W(1,1) = \{ V(1, 4 + 1 - 2), 20, 0) = \{8,15,4,20,0\}$

$W(1,2) = \{ V(1, 4 + 2 - 2), 12) = \{8,15,4,8,12\}$, etc.

The final schedule, which satisfies the system and capacity conditions, is:

Table 7.5. Final solution for the 5-period problem.

Stage	Period					
i	1	2	3	4	5	
1	(8,	15,	4,	8,	12)	[50]
2	(23,	0,	12,	0,	12)	[83]
3	(8,	19,	0,	8,	12)	[36]
	(39,	34,	16,	16,	36)	[169]

The feasible schedule for the 5-period problem at each stage is illustrated in the parentheses above, and the total cost associated with this plan is 169.

Conversely, the heuristic of this thesis was to produce as much of product item 1 as possible and produce product items 2 and 3 as required in period 1, then produce as much of product item 2 (provided $S_{1t} > d_{1t}$) as possible and produce product item 3 as required in period 2, and finally produce as much of product item 3 as possible (provided $S_{1t} > d_{1t}, S_{2t} > d_{2t}$) in period 3 and continue the same process for the rest of the periods. The details of the heuristic for the three-end-item problem were illustrated in chapter 4. The final plan using the heuristic of this thesis is:

Table 7.6. Solution using proposed heuristic.

Stage	Period					
i	1	2	3	4	5	
1	(23,	0,	24,	0,	0)	[67.0]
2	(8,	24,	0,	15,	0)	[86.0]
3	(8,	15,	4,	20,	0)	[44.0]
	(39,	39,	28,	35,	0)	[197.0]

The total cost in conjunction with the heuristic of this thesis is 197.0. This cost is greater than the cost obtained using Eftekharzadeh's heuristic and reflects the fact that the heuristic proposed in this thesis adopts a greedy approach to production by having few setups.

7.3. Sensivity Analysis of the Heuristics

A small example problem was illustrated in the previous section to show the relation between the two heuristics. According to the results, the Eftekharzadeh [1988] heuristic performed better than the heuristic of this thesis. This one result does not reflect the overall performance of his heuristic and the data structure was explored further to decide which heuristic is better under what circumstances. For that reason, the setup and holding costs were changed according to the following logic and the results are illustrated in Table 7.7.

Both heuristics, as will be seen from the final schedules, produce plans which are not determined by the costs, but are determined by the demand structure. In the following table different setup and holding costs were used with the same proportions in the given data. That is, $cs_1 = 1.25*cs_3$, $cs_2 = 2.5*cs_3$, and the same holding costs for each stage will be used to calculate the total costs in the following table. In Table 7.7, H1 refers to the heuristic of this thesis, and H2 refers to the total cost of Eftekharzadeh's heuristic.

Table 7.7. Total costs under different cost assumptions

Setup Cost	Holding Cost	H1	H2	Setup Cost	Holding Cost	H1	H2
3	1	127.0	80.3	8	0.1	120.5	144.7
4	1	141.0	98.0	8	0.2	129.0	147.4
5	1	155.0	115.8	8	0.3	137.5	150.1
6	1	169.0	133.5	8	0.4	146.0	152.8
7	1	183.0	151.3	8	0.5	154.5	155.5
8	1	197.0	169.0	8	0.6	163.0	158.2
9	1	211.0	186.7	8	0.7	171.5	160.9
10	1	225.0	204.5	8	0.8	180.0	163.6
11	1	239.0	222.3	8	0.9	188.5	166.3
12	1	253.0	240.0	8	1.0	197.0	169.0
13	1	267.0	257.8	8	1.1	205.5	171.7
14	1	281.0	275.5	8	1.2	214.0	174.4
15	1	295.0	293.3	8	1.3	222.5	177.1
16	1	309.0	311.0	8	1.4	231.0	179.8
17	1	323.0	328.8	8	1.5	239.5	182.5
18	1	337.0	346.5	8	1.6	248.0	185.2
19	1	351.0	364.3	8	1.7	256.5	187.9
20	1	365.0	382.0	8	1.8	265.0	190.6

In Table 7.7, the total costs in the third and fourth columns are calculated while the holding cost is held constant as before and the setup costs were allowed to vary. In addition, the setup cost was fixed at the same value while the holding costs varied. The results of the total costs of these changes are illustrated in the seventh and eight columns. The total cost of the heuristic of this thesis works well if the setup cost is

greater than or equal to the 16 times of the holding cost according to the above Table.

7.4. Conclusion

In summary: Eftekharzadeh's heuristic appears to works well if:

setup cost is low
holding cost is high (Setup cost/Holding cost $\leq$ 16)

conversely, the heuristic developed in this thesis appears to works well if;

setup cost is high
holding cost is low (Setup cost/Holding cost $\geq$ 16)

Furthermore, the size of the problem can also determine which heuristic is best. Eftekharzadeh stated that reporting results by '...Generating all possible schedules (optimal solution or complete enumeration) is not practical for large problems. This would require substantial CPU time.' On the other hand the heuristic developed in this thesis works according to very simple rules, and therefore handles large problems in a short time (see Tables 4.6.1-4.6.21). Therefore the size of the problem is not important for the heuristic developed in this thesis, because it requires very simple principles.

In addition, the heuristic of this thesis may be improved by shifting the production lots to left or right if possible as suggested by Karni and Roll [1982], Afentakis [1987]. This is one possible area for further research.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

8.1. Introduction

This thesis concentrated on using heuristics for the constrained product items with the EOQ and Silver-Meal approaches for the unconstrained product items for the multi-level lot-sizing problems when there is a bottleneck(s). Then, they are employed to the rolling schedule environment. It is illustrated that none of the optimal techniques are able to solve those problems.

This chapter will first summarise the research which was explained throughout the thesis, and finally some directions of future research will be provided.

8.2. Conclusions

A simple heuristic for the bottleneck multi-level lot-sizing problem has been developed. The following results are quoted from the experimental testing.

1. A heuristic for the multi-level lot-sizing problem was designed by grouping product items into (a) end-items, (b) non-end-items and two simple procedures were used independently. The reason for this categorisation is that production of each non-end-item outside the bottleneck is unconstrained and so has neither any affect on the production of any other non-end-item outside the bottleneck nor on the production of the items which are constrained by the bottleneck. The results illustrated that the heuristic solution for the constrained product

items with the EOQ and Silver-Meal approaches for the unconstrained product items were generally better than the integer programming solution of the Billington et al. [1986] model. It provided a great improvement in respect of the computational time and required only a small fraction of computer time required by the full integer programming approach. This heuristic was very easy to implement even without a computer routine. Thus simple operational procedures can be derived from the heuristic rules.

2. The heuristic for the constrained product items and the EOQ and Silver-Meal approaches for the unconstrained product items were applied to the rolling schedule situations to make stable plans in the case of uncertain demand cases. For this reason 96 period problems were investigated and the results showed that the total cost increased when the schedule involved the future demands.

3. The results of the heuristic solution for the constrained product items with the Silver-Meal approach for the unconstrained product items were better than the integer programming solution in all cases but the heuristic for the constrained product items with the EOQ approach for the unconstrained product items did not provide the same quality for a few cases (structure 2 for three-end-item problem, and structure 2 for one-end-item problem with 95% utilisation). The reason was that more setups were being used than necessary when the setup cost was above average.

4. The heuristic was applied to the assembly product structure to make some comparison between the heuristic of this thesis and Eftekharzadeh's heuristic. The results illustrated that the heuristic of this thesis performed well when the setup cost was greater than or equal to the

16 times of the holding cost according to the data given in section 7.3.

8.3. Recommendations for the Future Research

There are a number of extensions to this work that would provide interesting research topics. These include the following:

(i) The heuristic of this thesis may be used within a Genetic Algorithm which is a random search formed by taking any of the population (of solutions) randomly. This population could be any of the grouping for the five-end-item problems such as the production items (1, 2, 3), (4, 5). Then, a new population may be regenerated according to its fitness values. This regeneration continues as many times as desired until the minimum cost is reached. Solutions are normally represented in binary vector form to allow the genetic approach to operate. This would be possible when decisions as to production of a product in a given period are made (see Goldberg [1989]).

(ii) Simulated annealing may be applied to the multi-item lot-sizing problem when there is a bottleneck(s). The production items group (1, 2, 3), (4, 5) for five-end-item problems can be employed to generate the productions subject to demands and stocks for each group randomly in each period. This generation may be dependent on two rules: (1) if the stock is greater than demand, no production is proceeded, (2) if the stock is less than demand, the generation of production could be any number of the utilisation less the stock for each item. Then the total cost is calculated after generating all possible schedules. Total costs are calculated according to those schedules. If the value of the current objective function is less than (in case of minimisation problem) the

previous one, accept the solution. Otherwise accept the solution with a probability $P = e^{-d/T}$, where d is the increase in the objective function value, and T is the control mechanism (called temperature) which decreases monotonically with each successive iteration. This mechanism is defined in Reeves [1991], Kirkpatrick et al. [1983] and Osman and Potts [1989] in the following form:

$$T_i = T_{i-1} / (1 + g\ T_{i-1})$$

where g is the constant which is defined as the temperature falls from some initial value of T_o to a final value of T_f in f iteration. This is calculated using the following formula:

$$g = (T_o - T_f) / ((f - 1) T_o\ T_f)$$

(iii) Tabu Search may be used for the same problem. To apply the Tabu search, an initial solution is required. This initial solution may be the total cost of the production group (1, 2, 3),(4, 5) for five-end-item problems. Then it may search a neighbourhood of that solution for a better one by moving up and down except for a certain prohibited or 'tabu' set. This prohibited set may be subject to the integer programming solution or capacity condition. If the new solution is better than the initial one, it is dropped from the schedule and continues to search other alternatives. The stopping point for this approach may be the linear programming solution or some percentage of it (see Reeves [1991]).

In conclusion it can be seen that as the solution of many production scheduling problem is NP hard, the development of new heuristic methods to solve such problems continues whenever any promising general heuristic approach can be applied to these problems.

REFERENCES

Adam, E.E.Jr. and Ebert, R.J.E., [1989], Production and Operations Management, Concepts, Models and Behavior, Fourth edition, Prentice-Hall.

Afentakis, P., Gavish, B. and Karmarkar, U., [1984], 'Computationally Efficient Optimal Solutions to the Lot-Sizing Problem in Multistage Assembly Systems,' *Management Science*, 30(2), pp222-239.

Afentakis, P. and Gavish, B., [1986], 'Optimal Lot-Sizing Algorithms for Complex Product Structures,' *Operations Research*, 34(2), pp237-249.

Afentakis, P., [1987], 'A Parallel Heuristic Algorithm for Lot-Sizing in Multistage Production Systems,' *IIE Transactions*, 19(1), pp34-42.

Askin, R.G. and Rahavan, M., [1983], 'The Effect of Lot-Sizing on Workload Variablity,' *Journal of Operations Management*, 4(1), pp53-71.

Atkins, D.R. and Iyogun, P.O., [1988], 'A Heuristic with Lower Bound Performance Guarantee for the Multi Product Dynamic Lot-Size Problem,' *IIE Transactions*, 20(4), pp369-373.

Bahl, H.C. and Ritzman, L.P., [1984], 'A Cyclical Scheduling Heuristic for Lot Sizing with Capacity Constraints,' *International Journal of Production Research*, 22(5), pp791-800.

Bahl, H.C., Ritzman, L.P. and Gupta, J.N.D., [1987], 'Determining Lot Sizes and Resource Requirements: A Review,' *Operations Research*, 35(3), pp329-344.

Baker, K.R. and Peterson, D.W., [1979], 'An Analytical Framework for Evaluating Rolling Schedules,' *Management Science*, 25(4), pp341-351.

Baringer, R.L. and Fogarty, D.W., [1987], 'Joint Order Release Decisions under Dependent Demand,' *Production and Inventory Management*, First Quarter, pp55-61.

Berry, W.L., [1972], 'Lot Sizing Procedures for Requirements Planning Systems: A Framework for Analysis,' *Production and Inventory Management*, Second Quarter, pp19-34.

Biggs, J.R., Hahn, C.K. and Pinto, P.A., [1980], 'Performance of Lot-Sizing Rules in an MRP System with Different Operating Conditions,' *Academy of Management Review*, 5(1), pp89-96.

Billington, P.J., McClain, J.O. and Thomas, L.J., [1983], 'Mathematical Programming Approaches to Capacity-Constrained MRP Systems: Review, Formulation and Problem Reduction,' *Management Science*, 29(10), pp1126-1141.

Billington, P.J., [1983], 'Multi-Level Lot-Sizing with a Bottleneck Work Center, Ph.D. Dissertation, Cornell University.

Billington, P.J., McClain, J.O., and Thomas, L.J., [1986], 'Heuristic for Multilevel Lot-Sizing with a Bottleneck,' *Management Science*, 32(8), pp989-1006.

Billington, P.J., Blackburn, J.D., Maes, J., Millen, R.A. and Wassenhove, L.W.V., [1988], 'Multi-Product Scheduling in Multi-Stage Serial Systems,' In A. Chikan and M. C. Lovell (*eds*), The Economics of Inventory Management, Elsevier Science, Amsterdam.

Blackburn, J.D. and Millen, R.A., [1979], 'Selecting a Lot-Sizing Technique for a Single-Level Assembly Process: Part I-Analytical Results' *Production and Inventory Management*, Third Quarter, pp42-52.

Blackburn, J.D. and Millen, R.A., [1979], 'Selecting a Lot-Sizing Technique for a Single-Level Assembly Process: Part II-Emprical Results' *Production and Inventory Management*, Fourth Quarter, pp41-52.

Blackburn, J.D. and Millen, R.A., [1980], 'Heuristic Lot-Sizing Performance in a Rolling-Schedule Environment,' *Decision Sciences*, 11(4), pp691-701.

Blackburn, J.D. and Millen, R.A., [1982], 'The Impact of Rolling Schedule in a Multi-Level MRP Systems, *Journal of Operations Management*, 2(2), pp125-135.

Blackburn, J.D. and Millen, R.A., [1982], 'Improved Heuristics for Multi-Stage Requirements Planning Systems,' *Management Science*, 28(1), pp44-56.

Blackburn, J.D. and Millen, R.A., [1984], 'Simultaneous Lot-Sizing and Capacity Planning in Multi-Stage Assembly Processes,' *European Journal of Operational Research*, 16(1), pp84-93.

Browne J., Harhen, J. and Shivnan, J., [1988], Production Management Systems A CIM Perspective, Addison-Wesley Publishers Ltd., Cornwall, Great Britain.

Brown, R.G., [1967], Decision Rules for Inventory Management, Holt, Reinhart and Winston, New York.

Chae, W.S., [1988], 'A Heuristic Lot-Sizing/Scheduling Methodology of a Multi-Stage Capacitated Production System by Using Load Families,' PhD Thesis, The University of Wisconsin-Meddison.

Carlson, R.C., Jucker, J.V. and Kropp D.H., [1979], 'Less Nervous MRP Systems: A Dynamic Economic Lot-Sizing Approach,' *Management Science*, 25(8), pp754-761.

Chand, S. [1982], 'A Note on Dynamic Lot Sizing in a Rolling-Horizon Environment,' *Decision Sciences*, 13(1), pp113-119.

Collier, D.A., [1980], 'A Comparison of MRP Lot-Sizing Methods Considering Capacity Change Costs,' *Journal of Operations Management*, 1(1), pp23-29.

Crowston, W.B., Wagner, M.H. and Henshaw, A., [1972], 'A Comparison of Exact and Heuristic Routines for Lot-Size Determination in Multi-Stage Assembly Systems,' *IIE Transactions*, 4(4), pp313-317.

Crowston, W.B., Wagner, M.H. and Williams, J.F., [1973], 'Economic Lot Size Determination in Multi-stage Assembly Systems,' *Management Science*, 19(5), pp517-527.

Dixon, P.S. and Silver, E.A., [1981], 'A Heuristic Solution Procedure for Multi-Item Single Level, Limited Capacity, Lot Sizing Problem,' *Journal of Operations Management*, 2(1), pp23-29.

Dogramaci, A., Panaylotopoulos, J.E. and Adam, N.R., [1981], 'The Dynamic Lot Sizing Problem for Multiple

Items under Limited Capacity,' *AIIE Transactions*, 13(4), pp294-303.

Eisenhut, P.S., [1975], 'A Dynamic Lot Sizing Algorithm with Capacity Constraints,' *AIIE Transactions*, 7(2), pp170-176.

Eftekharzadeh, R.S.M., [1988], 'Multi-Stage Capacitated Lot-Sizing for Assembly Structure Manufacturing Systems: Two New Heuristics,' Working Paper, St. John's University, Department of Quantitative Analysis, New York, USA.

Fisher, M.L., [1980], 'Worst-Case Analysis of Heuristic Algorithm,' *Management Science*, 26(1), pp 1-17.

Florian, M. and Klein, M., [1971], 'Deterministic Production Planning with Concave Costs and Capacity Constraints,' *Management Science*, 18(1), pp 12-20.

Fogarty, D.W. and Barringer, R.L., [1988], 'Joint Order Release Decisions Under Dependent Demand,' In A. Chikan and M.C.Lovell (*eds*) The Economics of Inventory Management, Elsevier Science, Amsterdam.

Gabbay, H., [1979], ' Multi-Stage Production Planning,' *Management Science*, 25(11), pp1138-1148.

Garey, M.R. and Johnson, D.S., [1979], Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, San Francisco.

Goldberg, D.E., [1989], Genetic Algorithms in Search, Optimisation and Machine Learning, The University of Alabama, Addison-Wesley Publishing Company Inc., Canada.

Gorham, T., [1970], 'Dynamic Order Quantities,' *Production and Inventory Management*, First Quarter, pp75-81.

Goyal, S.K. and Gunasekaran, A., [1990], 'Invited Reviews, Multi-Stage Production-Inventory Systems,' *European Journal of Operational Research*, 46, pp1-20.

Gupta, Y.P. and Keung, Y., [1990], 'A Review of Multi-Stage Lot-Sizing Models,' *International Journal of Production and Management*, 10(9), pp57-73.

Hum, S.H., [1988], 'Integrated Production-Mixed Planning, Lotsizing and Scheduling of Bottleneck Facilities,' PhD, University of Calafornia, Los Angales.

Iyogun, P., [1991], 'Heuristic Methods for the Multi-Product Dynamic Lot Size Problem,' *Journal of Operational Research* , 42(10), pp889-894.

Jensen, P.A. and Khan, H.A., [1972], 'Scheduling in a Multi-Stage Production System with Setup and Inventory Costs,' *AIIE Transactions*, 4(3), pp126-133.

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

Karni, R. and Roll, Y., [1982], 'A Heuristic Algorithm for the Multi-Item Lot-Sizing Problem with Capacity Constraints,' *IIE Transactions*, 14(4), pp249-256.

Kirkpatrick, S., Jelatt, C.D. and Vecchi, M.P., [1983], 'Optimisation by Simulated Annealing,' *Science*, 220(4598), pp671-679.

Kropp, D.H. and Carlson R.C., [1984], 'A Lot-Sizing Algorithm for Reducing Nervousness in MRP Systems,' *Management Science*, 30(2), pp240-244.

Lambrecht, M.R. and Vander Eecken, J., [1978], 'A Facilities in Series Capacity Constrained Dynamic Lot-Size Model,' *European Journal of Operational Research*, 2(2), pp42-49.

Lambrecht, M.R. and Vanderveken, H., [1979], 'Heuristic Procedure for the Single Operation, Multi-Item Loading Problem,' *AIIE Transactions*, 11(4), pp 319-326.

Love, S.F., [1972], 'A Facilities in Series Inventory Model with Nested Schedules,' *Management Science*, 18(5), pp327-338.

Maes, J. and Wassenhove, L.N.V., [1986], 'A Simple Heuristic for the Multi-Item Single Level Capacitated Lot Sizing Problem,' *Operations Research Letters*, 4(6), pp265-273.

Maes, J., McClain, J.O. and Wassenhove, L.N.V., [1991], 'Multilevel Capacitated Lot Sizing Complexity and LP-Based Heuristics,' *European Journal of Operational Research*, 57(2), pp131-148.

Maes, J. and Wassenhove, L.N.V., [1991], 'Capacitated Dynamic Lot Sizing Heuristics for Serial Systems,' *International Journal of Production Research*, 29(6), pp1235-1249.

Manne, A.S., [1958], 'Programming of Economic Lot Sizes,' *Management Science*, 4(2), pp115-135.

Maxwell, W.L., Muckstadt, J.A., Thomas, L.J. and VanderEecken, J., [1983], 'A Modelling Framework for Planning and Control of Production in Discrete Parts Manufacturing and Assembly Systems,' *Interfaces*, 13(6), pp92-104.

Maxwell, W.L. and Muckstadt, J.A., [1985], 'Establishing Consistent and Realistic Reorder Interval in Production-Distribution Systems,' *Operations Research*, 33(6), pp1316-1341.

McCelland, M.K., Southard, M.H. and Wagner, H.M., [1988], 'Inventory and Lot-Size Strategies in an MRP Environment ,' In A. Chikan and M. C. Lovell, (*eds*), The Economics of Inventory Management, Elsevier Science Publishers B. V., Netherlands.

McClain, J.O. and Trigeiro, W.W., [1985], 'Cyclic Assembly Schedules,' *IIE Transactions*, 17(4), pp346-353.

McClain, J.O. and Thomas, L.J., [1980], Operations Management, Production of Goods and Services, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

McClain, J.O., Thomas, L.J. and Weiss, E.N., [1989], 'Efficient Solutions A Linear Programming Model for Production Scheduling with Capacity Constraints and No Initial Stock,' *IIE Transactions*, 21(2), pp144-152.

McLaren, B.T., [1977], 'A Study of Multiple Level Lot Sizing Procedures for Material Requirements Planning Systems,' PhD Thesis, Purdue University.

MGG User Guide, [1987], SD-Scicon, Milton Keynes, England.

Monks, J.G., [1987], Operations Management, Theory and Problems, Third Edition, McGraw-Hill, Management Series, America.

Osman, I.H. and Potts, C.N., [1989], 'Simulated Annealing for Permutation Flow-Shop Scheduling,' *OMEGA*, 17, pp551-557.

Orlicky, J., [1975], Material Requirements Planning, The New Way of Life in Production and Inventory Management, McGraw-Hill, New York.

Plossl, G.W. and Obec, W.W., [1975], Production and Inventory, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Ramsay, T.E.Jr., [1980], 'Integer Programming Approaches to Capacitated Concave Cost Production Planning Problems,' Ph.D Dissertatic Georgia Institute of Technology.

Reeves, C., [1991], 'An Introduction to Genetic Algorithms,' First O.R. Conference, Univeristy of Salford.

Rosenblatt, M.J., [1985], 'Fixed Cycle, Basic Cycle and EOQ Approaches to the Multi-Item Single-Supplier System,' *International Journal of Production Research*, 23(6), pp1131-1139.

Sciconic User Guide, [1986], SD-Scicon, Milton Keynes, England.

Schwarz, L.B. and Schrage, L., [1975], 'Optimal and System Myopic Policies for Multi-Echelon Production-

Inventory Assembly Systems,' *Management Science*, 21(11), pp1285-1294.

Silver, E.A. and Meal, H.C., [1973], 'A Heuristic for Selecting Lot Size Quantities for Deterministic Time-Varying Demand Rate and Discrete Opportunities for Replenishment,' *Production and Inventory Management*, 14(2), pp64-74.

Steinberg, E. and Napier, H.A., [1980], 'Optimal Multi-Level Lot Sizing for Requirements Planning Systems,' *Management Science*, 26(12), pp1258-1271.

Taha, H.A. and Skeith, R.W., [1970], 'The Economic Lot Sizes in Multi-Stage Production Systems,' *AIIE Transactions*, 2(2), pp157-162.

The Numerical Algorithms Group Limited, [1990] NAG Fortran Library Manual, Mark 14, Oxford.

Toklu, B. and Wilson, J.M., [1991], 'A Heuristic for Multilevel Lot-Sizing Problems with Multiple Bottleneck,' *Loughborough University Management Research Series*, Paper 1991:14.

Toklu, B. and Wilson, J.M., [1991], 'An Analysis of Multiple Bottleneck Problems in Multilevel Lot-Sizing Problems,' Operational Research Society Young Researcher's Conference, University of Salford.

Toklu, B. and Wilson, J.M., [1992], 'A Heuristic for Multi-Level Lot-Sizing Problems with a Bottleneck,' *International Journal of Production Research*, 30(4), pp787-798.

Wagner, H. and Whitin, T., [1959], 'Dynamic Version of the Economic Lot Size Model,' *Management Science*, 5, pp89-96.

Zahorik, A., Thomas, J. and Triegeiro, W.W., [1984], 'Network Programming Models for Production Scheduling in Multi-Stage, Multi-Item Capacitated System,' *Management Science*, 30(3), pp308-325.

Zangwill, W.I., [1969], 'A Backlogging Model and a Multi-Echelon Model of a Dynamic Economic Lot Size Production System-A Network Approach,' *Management Science*, 15(9), pp506-527.

APPENDIX A
Model in MGG on SCICONIC

```
OPTIONS OLDFORMAT
NOTATION
SUFFICES
 I IMAX 25
 K KMAX 25
 T TMAX 12
 L LMAX 12
VARIABLES
 P(I,T) '*IITT'
 X(I,T) '*IITT'
BOUND BV
EXTERNAL VALUES
 H(I) F5.2
 A(K,I) F5.1
 D(I,T) F5.1
 OI(I) F5.1
 B(I) F5.1
 S(I) F5.1
 CS(I) F6.1
 CAP(T) F7.1
 LA(K) I5
PROBLEM
MINIMISE
*TCOST '*****'
 SUM(I,T)A01*P(I,T)+SUM(I,T)B01*X(I,T)
SUBJECT TO
*CPRD '****KKLL' NOT IF (L.LE.LA(K))
 SUM(I,T)A02*P(I,T).GE.B02
*CAP '***TT'
 SUM(I)B03*P(I,T)+SUM(I)B04*X(I,T).LE.B05
*LIM '***IITT'
 P(I,T)-B06*X(I,T).LE.0.0
ELEMENTS
      B01=CS(I)
      B03=B(I)
      B04=S(I)
      B05=CAP(T)
      A01=H(I)*(TMAX-T+1)
      A02=Z02()
      B02=Z03()
      B06=199.0
FUNCTIONS
      FUNCTION Z02()
      Z02=0.0
      IF(T.GT.L)RETURN
      IF(I.GT.K)RETURN
      IF(I.EQ.K)GOTO100
      Z02=-A(K,I)
      RETURN
100   N=L-LA(K)
      IF(T.GT.N)RETURN
      Z02=1.0
      RETURN
      END
      FUNCTION Z03()
```

```
      Z03=-OI(K)
      DO 100 N=1,L
100   Z03=Z03+D(K,N)
      RETURN
      END
ENDATA
```

Sample Run Stream

```
1.    Sciconic
2.    Infile='Matrix.d'
3.    Convert
4.    Setup
5.    Primal
6.    Global
7.    Prints
8.    Stop
```

1. This is the macro command to start the software.
2. This identifies the file which MGG has produced.
3,4. These set the problem up and organise the matrix.
5. This solves the linear programming (LP) problem.
6. This solves the integer programming (IP) problem.
7. This prints out results.
8. This ends the software.

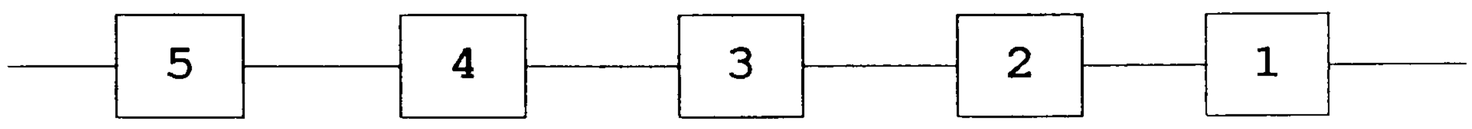
APPENDIX B
Data

One End Item Bottleneck Problem

Randomly Generated Cost Sets

	Cost Set 1:		Cost Set 2:		Cost Set 3:	
i	h_i	S_i	h_i	Si	h_i	S_i
1	0.5	300	0.5	500	0.1	300
2	0.1	200	0.5	400	2.0	500
3	0.5	200	2.0	400	0.1	500
4	0.1	500	1.0	200	1.0	300
5	1.0	400	1.0	200	1.0	500

All bi = 1.0 for the item made on the work centre.
All aij = 1.0 for all predecessor relations shown below:



One End Item Bottleneck Problem

Randomly Generated Demand Stream

Low Coefficient of Variation, C_v = 0.1057

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	33	43	36	46	46	42	38	41	44	40	35	46

Medium Coefficient of Variation C_v = 0.1862

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	54	47	80	63	45	76	74	55	54	58	76	58

High Coefficient of Variation C_v = 0.3464

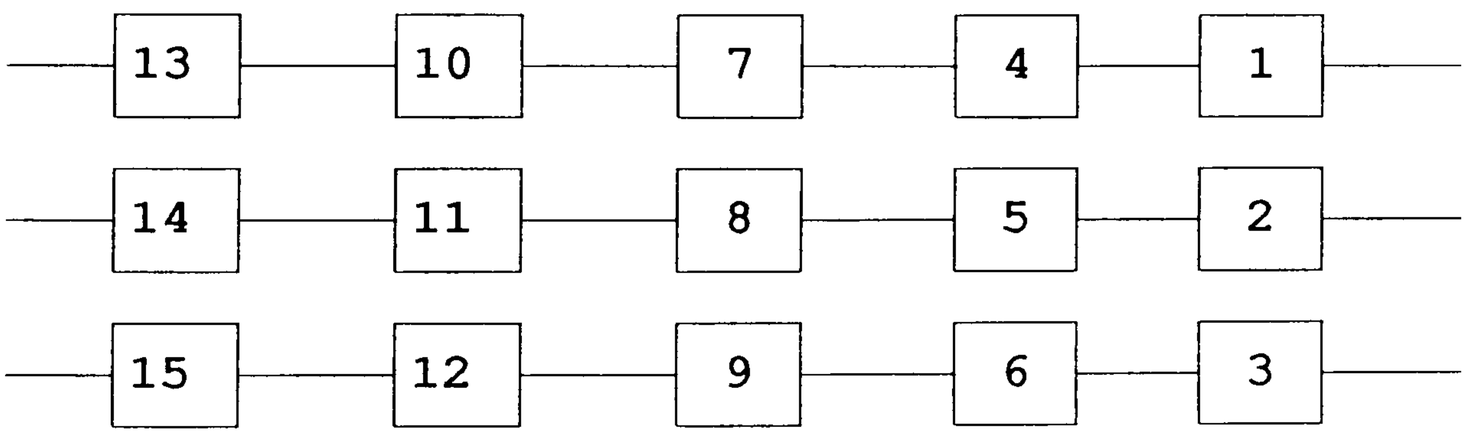
	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	17	31	58	51	64	42	44	18	48	41	58	32

Three End Item Bottleneck Problem

Randomly Generated Cost Sets

	Cost Set 1:		Cost Set 2:		Cost Set 3:	
i	h_i	S_i	h_i	Si	h_i	S_i
1	0.5	500	2.0	500	0.5	400
2	0.5	400	2.0	300	1.0	300
3	1.0	300	0.1	300	2.0	300
4	0.1	200	2.0	400	0.1	300
5	1.0	400	2.0	500	1.0	500
6	0.1	500	1.0	400	0.1	200
7	2.0	300	2.0	400	2.0	500
8	1.0	300	2.0	200	0.5	500
9	2.0	300	2.0	500	0.1	300
10	0.5	500	0.1	300	1.0	300
11	2.0	400	1.0	200	0.1	400
12	0.1	400	0.5	300	0.5	500
13	1.0	200	1.0	300	1.0	500
14	1.0	200	1.0	500	1.0	300
15	1.0	500	1.0	300	1.0	400

All bi = 1.0 for the item made on the work centre.
All aij = 1.0 for all predecessor relations shown below:



Three End Item Bottleneck Problem

Randomly Generated Demand Stream

Low Coefficient of Variation, C_v = 0.1144

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	41	59	69	47	41	73	65	44	63	52	79	55
2	56	55	54	65	76	72	42	70	76	50	45	74
3	28	35	29	51	40	44	42	45	52	38	49	40

Medium Coefficient of Variation C_v = 0.1926

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	23	33	49	32	13	34	50	50	09	23	25	21
2	56	50	24	35	77	35	42	72	50	71	41	25
3	35	35	31	77	54	17	83	41	60	53	44	76

High Coefficient of Variation C_v = 0.3955

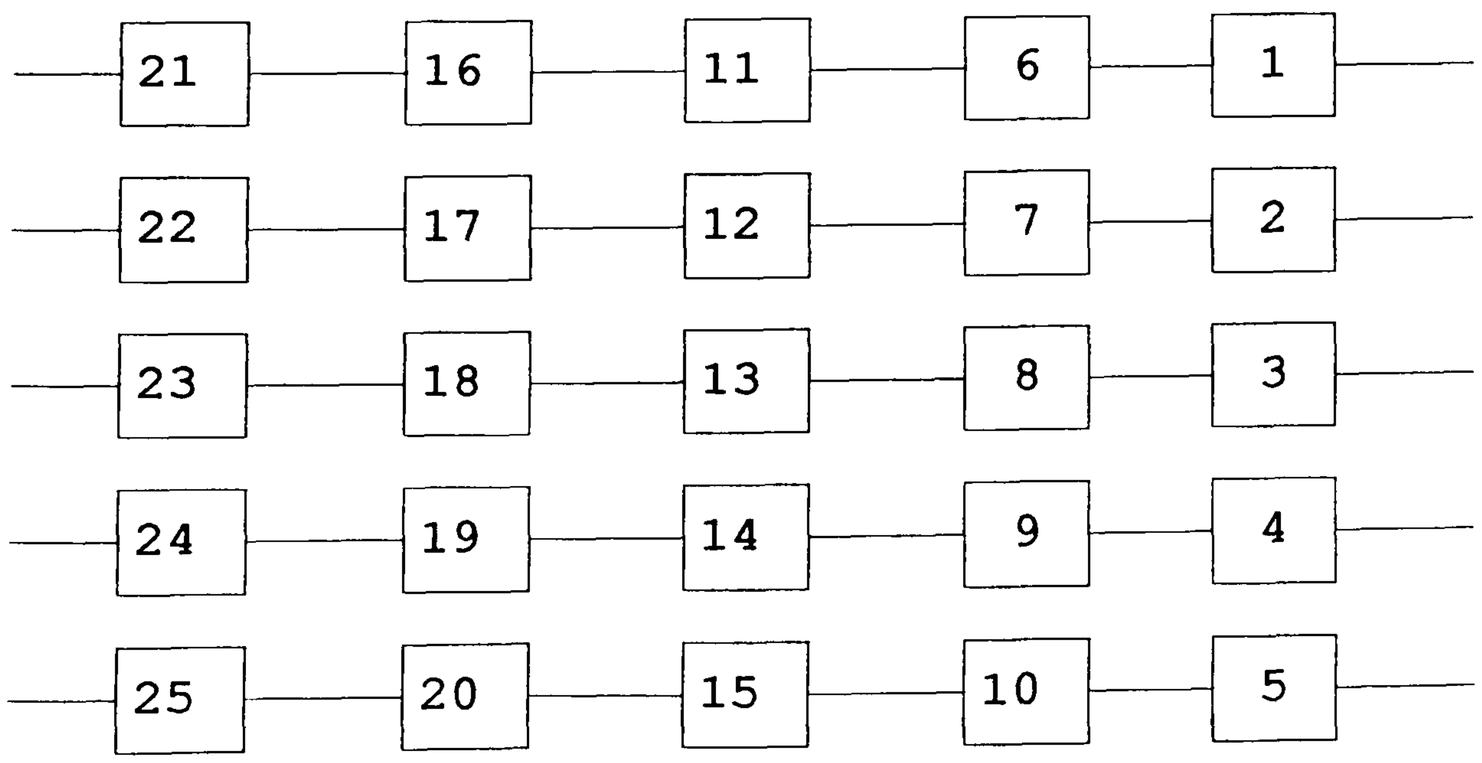
	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	00	07	21	01	41	65	47	54	00	63	43	62
2	00	117	126	93	55	105	08	126	82	45	18	93
3	25	00	14	17	29	56	105	99	96	42	58	00

Five End Item Bottleneck Problem

Randomly Generated Cost Sets

	Cost Set 1:		Cost Set 2:		Cost Set 3:	
i	h_i	S_i	h_i	Si	h_i	S_i
1	0.1	200	1.0	200	0.1	200
2	0.5	500	0.5	300	0.1	500
3	2.0	200	0.1	200	0.1	400
4	2.0	300	0.5	300	2.0	400
5	1.0	300	0.1	400	2.0	400
6	2.0	300	2.0	500	2.0	500
7	0.5	300	0.5	200	1.0	200
8	1.0	500	2.0	300	0.1	500
9	1.0	500	0.1	300	0.5	500
10	2.0	200	1.0	400	1.0	200
11	0.1	400	2.0	500	0.1	200
12	0.1	400	1.0	200	1.0	400
13	0.1	300	2.0	200	0.1	500
14	2.0	300	0.1	500	2.0	300
15	0.1	500	0.1	500	0.1	400
16	1.0	400	2.0	300	0.5	200
17	2.0	500	0.5	500	0.1	300
18	2.0	500	0.1	500	2.0	400
19	0.1	500	2.0	200	0.5	200
20	1.0	300	0.1	300	1.0	200
21	1.0	200	1.0	400	1.0	400
22	1.0	300	1.0	300	1.0	400
23	1.0	500	1.0	400	1.0	400
24	1.0	200	1.0	500	1.0	500
25	1.0	400	1.0	200	1.0	500

All bi = 1.0 for the item made on the work centre.
All aij = 1.0 for all predecessor relations shown below:



Five End Item Bottleneck Problem

Randomly Generated Demand Stream

Low Coefficient of Variation, C_v = 0.0790

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	29	64	59	46	82	86	74	36	36	59	86	54
2	38	47	52	37	44	25	52	34	27	37	34	50
3	51	27	35	30	43	55	55	60	50	43	48	29
4	33	42	44	77	43	31	53	33	53	51	71	48
5	74	84	67	89	56	46	55	73	59	74	42	75

Medium Coefficient of Variation C_v = 0.2125

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	43	04	93	111	12	70	04	113	72	122	35	105
2	13	60	14	46	22	46	33	39	47	54	33	09
3	58	33	61	00	67	71	77	00	50	76	112	00
4	20	65	13	33	13	81	06	53	42	57	66	00
5	28	79	32	54	100	14	83	01	22	28	01	33

High Coefficient of Variation C_v = 0.4282

	Period											
i	1	2	3	4	5	6	7	8	9	10	11	12
1	27	110	00	00	151	139	72	41	10	31	87	102
2	76	15	50	36	00	19	24	00	40	92	83	114
3	11	07	21	97	87	87	00	82	105	00	79	23
4	12	00	02	93	22	78	29	00	00	118	109	02
5	70	00	00	100	79	62	46	00	10	36	00	18

APPENDIX C
Heuristic Program on Fortran 77

```
c       This program calculates the total cost for multi
c       level lot-sizing problem with bottleneck(s)
c
        program bottleneck
c
c
        integer t,blkstart,blkend
        character*14 filename,name1,cname2
c
c
          dimension prod(25,5,96),stock(25,5,96),aver(5,3)
       1,hi(25),cs(25),it1(96),d(5,96),ibp(25),sumh(25)
       2,sum1(25),sumn(25),sums(25),dem(5,96),c(25),h(25)
       3,dema(5,96),cap(25),stock1(25,5,96),stock2(25,5,96)
       4,iutd(25),stock3(25,5,96),scost(25,5,96)
       5,prod1(25,5,96),stock4(25,5,96),prod2(25,5,96)
       6,sumsil(25)
c
c
          data (iutd(i),i=1,25)/25*0/
c
c
          print*,'blksize=,blkjump=,periods='
          read*,blksize,blkjump,periods
          print*,'give the first sub-program name please'
c
c
c       Now reading the first sub-program name. If the
c       first sub-program name which is ' name1 ' is equal
c       to 'roll', calculation is done using rolling
c       schedule. Otherwise it is done without rolling
c       schedule for period 1 to 96.
c
c
          read*,name1
c
c
          print*,'give the second sub-program name  please'
c
c
c       now reading the second sub- program name. If the
c       second sub-program name is 'silver', calculation
c       is executed using silver meal technique.Otherwise
c       it is executed using economic order quantity from
c       period 1 to 96.
c
c
          read*,cname2
          print *,'give data file name please'
c
c
c       now reading data from files
c

c
```

```
      read *,filename
      open(unit=7,file=filename,status='old')
      open(unit=9,file="output")
      write(9,169)
c
c
c     'ibottle' is the number of bottleneck', and
c     'ibp(j)' is the position of the bottleneck, 'd' is
c     the demand, 'aver'is the total or average demandin
c     accordance with the index, 'hi' is the holding
c     cost, 'cs' is the setup cost, 'prod' is the amount
c     of production, 'stock' is the  amount of the
c     available stock, 'cap' is the capacity which is
c     used,'icho' is the number of end items, 'iutd' is
c     the capacity utilisation,'kbt' is the number of
c     bottleneck(s),'lk' is the number of end item(s)for
c     five-end item problems, 'k' is the number of
c     item(s) for one or three end item problems
c
c
c     Now reading the end items, bottleneck and capacity
c     utilisation.
c
c
      read*,icho,ibottle
      do 1 j=1,ibottle
        write(9,94)
        read *,ibp(j),cap(j)
1     continue
      read(7,*)((d(i,j),j=1,PERIODS),i=1,icho)
      read(7,*)(cs(i),i=1,5*icho)
      read(7,*)(hi(i),i=1,5*icho)
c
c
c     NOW STARTING THE CALCULATIONS
c
c
c     BLOCKSTART IS EQUAL TO BLOCK LOOP
c
c
      total=0.0
      do 7 blkstart=1,periods,blkjump
        blkend=blkstart+blksize-1
        if(blkend.gt.periods) goto 999
        do 2 i=1,icho
          do 2 j=1,3
            aver(i,j)=0.0
2       continue
        sum=0.0
        do 4 j=1,icho
          do 3 t=blkstart,blkend
            do 111 lk=1,25
              stock(lk,j,t)=0.0
              stock1(lk,j,t)=0.0
              stock2(lk,j,t)=0.0
```

```
                  stock3(lk,j,t)=0.0
                  prod(lk,j,t)=0.0
                  scost(lk,j,t)=0.0
                  prod1(lk,j,t)=0.0
                  stock4(lk,j,t)=0.0
111             prod2(lk,j,t)=0.0
                dem(j,t)=0.0
                dema(j,t)=0.0
                aver(j,1)=aver(j,1)+d(j,t)
3             continue
                aver(j,2)=int(aver(j,1)/blksize+0.5)
                sum=sum+aver(j,1)
4             continue
              sumt=0.0
              kbt=1
              icstep=1
              if(icho.eq.1)then
                i1=1
                i2=5
              elseif(icho.eq.3)then
                i1=1
                i2=13
              elseif(icho.eq.5)then
                i1=1
                i2=21
                icstep=3
              endif
                  do 41 lk=i1,i2,icho
                    do 8 ich=1,icho,icstep
                      k=lk+ich-1
                      if(k.eq.ibp(kbt))then
                        iutd(lk)=int(sum/(cap(kbt)*blksize)
     1+0.99)
                        write(9,93)cap(kbt),iutd(lk)
                        write(9,99)((aver(i,j),j=1,2),i=1,
     1icho),sum
                        if(icho-3)15,20,30
c
c
c       'ANAL' IS THE MAIN PROGRAM FOR ONE END ITEM
        PROBLEM
c
c
15      call anal(iutd,d,prod,stock,aver,blkstart,blkend,
     1ich,lk,k,name1)
        write(9,961)((prod(lk,i,j),i=1,icho),
     1,j=blkstart,blkend)
        write(9,110)((stock(lk,i,j),i=1,icho),
     1j=blkstart,blkend)
110     format('stock',1(2x,f9.0))
961     format('production',1(2x,f9.0))
        go to 40
c
c
```

```
c       'ANAL1' IS THE MAIN PROGRAM FOR THREE END ITEM
c       PROBLEM
c
20      call anal1(iutd,d,prod,stock,aver,blkstart,blkend
     1,ich,lk,k,name1)
        write(9,98)((prod(lk,i,j),i=1,icho),j=blkstart
     1,blkend)
        write(9,97)((stock(lk,i,j),i=1,icho),j=blkstart
     1,blkend)
        go to 40
c
c
c
c   'ANAL2' IS THE MAIN PROGRAM FOR FIVE END ITEM PROBLEM
c
c
30      call anal2(kbt,iutd,d,prod,stock,aver,blkstart
     1,blkend,dem,lk,k,name1)
        write(9,130)((prod(lk,i,j),i=1,icho),j=blkstart
     1,blkend)
        write(9,140)((stock(lk,i,j),i=1,icho),j=blkstart
     1,blkend)
140     format('stock',5(2x,f9.0))
130     format('production',5(2x,f9.0))
        endif
c
c
40      if(cname2.ne.'silver')then
c
c
c       now calling the subroutine cost for non-end-items
c       for non-bottleneck item(s) using the eoq technique
c
c
        call cost(kbt,k,prod,stock,aver,d,hi,cs,ich
     1,ibottle,ibp,sumc,t,icho,blkstart,blkend,blksize
     2,sumh,suml,sumn,sums,dem,c,h,dema,lk,stock1,stock2
     3,stock3,name1)
        sumt=sumt+sumc
        total=total+sumc
c
c
        else
c
c
c       now calling the subroutine costs for non end-items
c       for non-bottleneck item(s) using silver-meal
c       technique
c
        call costs(kbt,k,prod,stock,aver,d,hi,cs,ich
     1,ibottle,ibp,sumc,t,icho,blkstart,blkend,blksize
     2,sumh,dem,c,h,dema,lk,stock1,stock2,stock3,scost
     3,prod1,stock4,prod2,sumsil,name1)
        sumt=sumt+sumc
        total=total+sumc
        endif
```

```
8       continue
41      write(9,95)k,sumt
        write(9,100)total
100     format(2x,'total=',f19.2)
93      format(' capacity utilization=',f7.5,5x,'capacity
     1,=',i5)
94      format(' give position of bottleneck & capacity')
95      format(i4,2x,'cost=',f17.1,3x)
169     format('please give no of end items & no of
     1bottlenecks')
97      format('stock',3(2x,f9.0))
98      format('production',3(2x,f9.0))
99      format(13f6.1)
7       continue
999     stop
        end
c
c
c       STARTS OF SUBROUTINE ANAL
c       FUNCTIONS: TO CALCULATE THE PRODUCTION AND STOCKS
c       FOR ONE-END ITEM PROBLEM FOR BOTTLENECK(S)
c       PARAMETERS: IUTD,D,PROD,STOCK,BLKSTART,BLKEND,LK,K
c
c
        subroutine anal(iutd,d,prod,stock,aver,blkstart
     1,blkend,ich,lk,k,name1)
c
c
        integer blkstart,blkend
        dimension iutd(25),d(5,96),prod(25,5,96)
     1,stock(25,5,96),aver(5,3)
c
c
        idif=0
        aver1=aver(1,1)
        if(blkstart.eq.1.or.name1.ne.'roll')then
          stock(lk,1,blkstart-1)=0.0
        endif
        idif=idif+stock(lk,1,blkstart-1)
        do 5 j=blkstart,blkend
          if(stock(lk,1,j-1).lt.d(1,j)) go to 6
            stock(lk,1,j)=stock(lk,1,j-1)-d(1,j)
            prod(lk,1,j)=0
            go to 5
6         if((aver1-idif).gt.iutd(lk))then
            prod(lk,1,j)=iutd(lk)
            stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
            idif=idif+prod(lk,1,J)
          else
            prod(lk,1,j)=aver1-idif
            stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
            idif=idif+prod(lk,1,j)
          endif
```

```
5       continue
        return
        end
c
c
c       START OF SUBROUTINE ANAL1
c       FUNCTIONS: TO CALCULATE THE PRODUCTION AND STOCKS
c       FOR THREE-END END ITEMS PROBLEM FOR BOTTLENECK(S)
c       PARAMETERS: IUTD,D,PROD,STOCK,LK,K,PERIODS
c
c
c
          subroutine anal1(iutd,d,prod,stock,aver,blkstart
      1,blkend,ich,lk,k,name1)
          integer blkstart,blkend
          dimension iutd(25),d(5,96),prod(25,5,96)
      1,stock(25,5,96),aver(5,3)
          idif1=0
          idif2=0
          idif3=0
          aver1=aver(1,1)
          aver2=aver(2,1)
          aver3=aver(3,1)
          if(blkstart.eq.1.or.name1.ne.'roll')then
            stock(lk,1,blkstart-1)=0.0
            stock(lk,2,blkstart-1)=0.0
            stock(lk,3,blkstart-1)=0.0
          endif
          idif1=idif1+stock(lk,1,blkstart-1)
          idif2=idif2+stock(lk,2,blkstart-1)
          idif3=idif3+stock(lk,3,blkstart-1)
          do 6 j=blkstart,blkend
            if(stock(lk,1,j-1).ge.d(1,j)) go to 13
              if(stock(lk,2,j-1).ge.d(2,j)) go to 7
                if(stock(lk,3,j-1).ge.d(3,j))go to 56
                  iprod=aver1+aver2+aver3-idif1-idif2-idif3
                  iprod1=aver1-idif1+d(2,j)-stock(lk,2,j-1
      1)+d(3,j)-stock(lk,3,j-1)
                  if(iprod.gt.iutd(lk).and.iprod1.gt
      1.iutd(lk))then
                    prod(lk,3,j)=d(3,j)-stock(lk,3,j-1)
                    stock(lk,3,j)=0.0
                    prod(lk,2,j)=d(2,j)-stock(lk,2,j-1)
                    stock(lk,2,j)=0.0
                    prod(lk,1,j)=iutd(lk)-prod(lk,2,j)-
      1prod(lk,3,j)
                    stock(lk,1,j)=prod(lk,1,j)-d(1,j)
      1+stock(lk,1,j-1)
                  elseif(iprod.lt.iutd(lk))then
                    prod(lk,3,j)=aver3-idif3
                    stock(lk,3,j)=prod(lk,3,j)-d(3,j)
      1+stock(lk,3,j-1)
                    prod(lk,2,j)=aver2-idif2
                    stock(lk,2,j)=prod(lk,2,j)-d(2,j)
      1+stock(lk,2,j-1)
                    prod(lk,1,j)=aver1-idif1
```

```
                stock(lk,1,j)=prod(lk,1,j)-d(1,j)
1+stock(lk,1,j-1)
              elseif(iprod1.le.iutd(lk))then
                prod(lk,3,j)=d(3,j)-stock(lk,3,j-1)
                stock(lk,3,j)=0.0
                prod(lk,2,j)=d(2,j)-stock(lk,2,j-1)
                stock(lk,2,j)=0.0
                prod(lk,1,j)=aver1-idif1
                stock(lk,1,j)=prod(lk,1,j)-d(1,j)
1+stock(lk,1,j-1)
              endif
                idif2=idif2+prod(lk,2,j)
                idif3=idif3+prod(lk,3,j)
                idif1=idif1+prod(lk,1,j)
                go to 6
7           stock(lk,2,j)=stock(lk,2,j-1)-d(2,j)
            prod(lk,2,j)=0.0
8           if(stock(lk,3,j-1).ge.d(3,j))go to 9
              iprod=aver1-idif1+aver3-idif3
              if(iprod.gt.iutd(lk))go to 55
                if((aver3-idif3).ge.iutd(lk))then
                  prod(lk,3,j)=d(3,j)-stock(lk,3,j-1)
                  stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
                else
                  prod(lk,3,j)=aver3-idif3
                  stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
                endif
                idif3=idif3+prod(lk,3,j)
                go to 58
9               stock(lk,3,j)=stock(lk,3,j-1)-d(3,j)
                prod(lk,3,j)=0.0
10            if((aver1-idif1).ge.iutd(lk))then
                prod(lk,1,j)=iutd(lk)
                stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
              else
                prod(lk,1,j)=aver1-idif1
                stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
              endif
              idif1=idif1+prod(lk,1,j)
              go to 6
58            iprod=aver1-idif1-prod(lk,2,j)-prod(lk,3,j)
              if(iprod.ge.iutd(lk))then
                prod(lk,1,j)=iutd(lk)-prod(lk,2,j)-
     1prod(lk,3,j)
                stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
              else
                prod(lk,1,j)=aver1-idif1
                stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
              endif
```

```
                idif1=idif1+prod(lk,1,j)
                go to 6
56              stock(lk,3,j)=stock(lk,3,j-1)-d(3,j)
                prod(lk,3,j)=0.0
                iprod=aver1-idif1+aver2-idif2
                iprod1=aver1-idif1+d(2,j)-stock(lk,2,j-1)
                if(iprod.gt.iutd(lk).and.iprod1.gt
     1.iutd(lk))then
                  prod(lk,2,j)=d(2,j)-stock(lk,2,j-1)
                  stock(lk,2,j)=0.0
                  prod(lk,1,j)=iutd(lk)-prod(lk,2,j)
                  stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
                elseif(iprod.le.iutd(lk))then
                  prod(lk,2,j)=aver2-idif2
                  stock(lk,2,j)=prod(lk,2,j)-d(2,j)
     1+stock(lk,2,j-1)
                  prod(lk,1,j)=aver1-idif1
                  stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
                elseif(iprod1.le.iutd(lk))then
                  prod(lk,2,j)=d(2,j)-stock(lk,2,j-1)
                  stock(lk,2,j)=0.0
                  prod(lk,1,j)=aver1-idif1
                  stock(lk,1,j)=prod(lk,1,j)-d(1,j)
     1+stock(lk,1,j-1)
                endif
                idif2=idif2+prod(lk,2,j)
                idif1=idif1+prod(lk,1,j)
                go to 6
13              stock(lk,1,j)=stock(lk,1,j-1)-d(1,j)
                prod(lk,1,j)=0.0
                if(stock(lk,2,j-1).ge.d(2,j)) go to 16
                  if(stock(lk,3,j-1).ge.d(3,j))go to 14
                    iprod=aver2-idif2+aver3-idif3
                    if(iprod.gt.iutd(lk))go to 50
                      if((aver3-idif3).ge.iutd(lk))then
                        prod(lk,3,j)=d(3,j)-stock(lk,3,j-1)
                        stock(lk,3,j)=stock(lk,3,j-1)
     1+prod(lk,3,j)-d(3,j)
                      else
                        prod(lk,3,j)=aver3-idif3
                        stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
                        endif
                        idif3=idif3+prod(lk,3,j)
                        go to 15
14                  stock(lk,3,j)=stock(lk,3,j-1)-d(3,j)
                    prod(lk,3,j)=0.0
15                  if((aver2-idif2).ge.iutd(lk))then
                      prod(lk,2,j)=iutd(lk)-prod(lk,3,j)
                      stock(lk,2,j)=prod(lk,2,j)-d(2,j)
     1+stock(lk,2,j-1)
                    else
                      prod(lk,2,j)=aver2-idif2
```

```
                stock(lk,2,j)=prod(lk,2,j)-d(2,j)
     1+stock(lk,2,j-1)
               endif
               idif2=idif2+prod(lk,2,j)
               go to 6
16           stock(lk,2,j)=stock(lk,2,j-1)-d(2,j)
             prod(lk,2,j)=0.0
             if(stock(lk,3,j-1).ge.d(3,j)) go to 17
               if((aver3-idif3).ge.iutd(lk))then
                 prod(lk,3,j)=iutd(lk)
                 stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
               else
                 prod(lk,3,j)=aver3-idif3
                 stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
               endif
               idif3=idif3+prod(lk,3,j)
               go to 6
50               prod(lk,3,j)=d(3,j)-stock(lk,3,j-1)
                 stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
                 idif3=idif3+prod(lk,3,j)
                 if((aver2-idif2+prod(lk,3,j)
     1).ge.iutd(lk))then
                   prod(lk,2,j)=iutd(lk)-prod(lk,3,j)
                   stock(lk,2,j)=prod(lk,2,j)-d(2,j)
     1+stock(lk,2,j-1)
                 else
                   prod(lk,2,j)=aver2-idif2
                   stock(lk,2,j)=prod(lk,2,j)-d(2,j)
     1+stock(lk,2,j-1)
                 endif
                 idif2=prod(lk,2,j)+idif2
                 go to 6
52                prod(lk,2,j)=iutd(lk)-d(3,j)
                  stock(lk,2,j)=prod(lk,2,j)-d(2,j)
     1+stock(lk,2,j-1)
                  idif2=idif2+prod(lk,2,j)
                  go to 6
55           prod(lk,3,j)=d(3,j)-stock(lk,3,j-1)
             stock(lk,3,j)=prod(lk,3,j)-d(3,j)
     1+stock(lk,3,j-1)
             idif3=idif3+prod(lk,3,j)
             iprod=aver1-idif1+prod(lk,2,j)+prod(lk,3,j)
             if(iprod.gt.iutd(lk))then
               prod(lk,1,j)=iutd(lk)-prod(lk,3,j)-
    1prod(lk,2,j)
               stock(lk,1,j)=prod(lk,1,j)-d(1,j)
    1+stock(lk,1,j-1)
           else
               prod(lk,1,j)=aver1-idif1
               stock(lk,1,j)=prod(lk,1,j)-d(1,j)
    1+stock(lk,1,j-1)
               endif
```

```
                idif1=prod(lk,1,j)+idif1
                go to 6
17            stock(lk,3,j)=stock(lk,3,j-1)-d(3,j)
              prod(lk,3,j)=0.0
6         continue
          return
          end
c
c
c         START OF SUBROUTINE ANAL2
c         FUNCTIONS: TO CALCULATE THE PRODUCTION AND STOCKS
c         FOR FIVE-END ITEMS PROBLEM FOR BOTTLENECK(S)
c
c
          subroutine anal2(kbt,iutd,d,prod,stock,aver
         1,blkstart,blkend,dem,lk,k,name1)
          integer blkstart,blkend
          dimension iutd(25),d(5,96),prod(25,5,96)
         1,stock(25,5,96),aver(5,3),dem(5,96)
          do 11 j=blkstart,blkend
            sum=0.0
            dem(1,j)=0.0
            dem(4,j)=0.0
            do 13 i=1,3
              sum=sum+d(i,j)
13          continue
            dem(1,j)=sum+dem(1,j)
            sum=0
            do 14 i=4,5
              sum=sum+d(i,j)
14          continue
11        dem(4,j)=sum+dem(4,j)
          idif1=0
          idif2=0
          aver1=aver(1,1)+aver(2,1)+aver(3,1)
          aver2=aver(4,1)+aver(5,1)
          if(blkstart.eq.1.or.name1.ne.'roll')then
            stock(lk,1,blkstart-1)=0.0
            stock(lk,4,blkstart-1)=0.0
          endif
          idif1=idif1+stock(lk,1,blkstart-1)
          idif2=idif2+stock(lk,4,blkstart-1)
          do 6 j=blkstart,blkend
            if(stock(lk,1,j-1).ge.dem(1,j)) go to 20
              if(stock(lk,4,j-1).ge.dem(4,j)) go to 21
                iprod=aver1+aver2
                if((iprod-idif1-idif2).gt.iutd(lk))then
                  prod(lk,4,j)=dem(4,j)-stock(lk,4,j-1)
                  stock(lk,4,j)=0
                  prod(lk,1,j)=iutd(lk)-prod(lk,4,j)
                  stock(lk,1,j)=prod(lk,1,j)-dem(1,j)
     1+stock(lk,1,j-1)
                else
                  prod(lk,4,j)=aver2-idif2
                  stock(lk,4,j)=prod(lk,4,j)-dem(4,j)
```

```
1+stock(lk,4,j-1)
            prod(lk,1,j)=aver1-idif1
            stock(lk,1,j)=prod(lk,1,j)-dem(1,j)
1+stock(lk,1,j-1)
          endif
            idif1=idif1+prod(lk,1,j)
            idif2=idif2+prod(lk,4,j)
              go to 6
21          stock(lk,4,j)=stock(lk,4,j-1)-dem(4,j)
            prod(lk,4,j)=0
            if((aver1-idif1).gt.iutd(lk))then
              prod(lk,1,j)=iutd(lk)
              stock(lk,1,j)=prod(lk,1,j)-dem(1,j)
      1+stock(lk,1,j-1)
            else
              prod(lk,1,j)=aver1-idif1
              stock(lk,1,j)=prod(lk,1,j)-dem(1,j)
      1+stock(lk,1,j-1)
            endif
            idif1=idif1+prod(lk,1,j)
            go to 6
20        stock(lk,1,j)=stock(lk,1,j-1)-dem(1,j)
          prod(lk,1,j)=0
          if(stock(lk,4,j-1).ge.dem(4,j)) go to 35
            if((aver2-idif2).gt.iutd(lk))then
              prod(lk,4,j)=iutd(lk)
              stock(lk,4,j)=prod(lk,4,j)-dem(4,j)
      1+stock(lk,4,j-1)
            else
              prod(lk,4,j)=aver2-idif2
              stock(lk,4,j)=prod(lk,4,j)-dem(4,j)
      1+stock(lk,4,j-1)
            endif
            idif2=idif2+prod(lk,4,j)
            go to 6
35        stock(lk,4,j)=stock(lk,4,j-1)-dem(4,j)
          prod(lk,4,j)=0
6         continue
          if(kbt.gt.1)goto 8
            do 37 j=1,2
              if(j.eq.2)then
                j1=4
                j2=5
              else
                j1=j
                j2=j+2
          endif
          sum=0.0
          sumt=0.0
          sumt1=0.0
          sum1=0.0
          do 36 i=j1,j2
            sum=sum+aver(i,2)
            sumt=sumt+aver(i,1)
36        continue
```

```
                aver(j1,2)=sum
                aver(j1,1)=sumt
37            continue
8           return
            end
c
c
c           START OF SUBROUTINE COST
c           FUNCTIONS: TO CALCULATE THE COST(S) FOR NON-END
c           ITEM(S) PROBLEM USING ECONOMIC ORDER QUANTITY
c
c
              subroutine cost(kbt,k,prod,stock,aver,d,hi,cs
     1,ich,ibottle,ibp,sumc,t,icho,blkstart,blkend
     2,blksize,sumh,suml,sumn,sums,dem,c,h,dema,lk,stock1
     3,stock2,stock3,name1)
              integer blkstart,blkend
              dimension prod(25,5,96),stock(25,5,96),aver(5,3)
     1,hi(25),cs(25),itl(96),d(5,96),ibp(25),sumh(25)
     2,suml(25),sumn(25),sums(25),dem(5,96),c(25),h(25)
     3,dema(5,96),cap(25),stock1(25,5,96),stock2(25,5,96)
     4,iutd(25),stock3(25,5,96)
c
c
c             NOW STARTING THE CALCULATION OF HOLDING AND SETUP
c             COST(S) FOR FIVE-END ITEM PROBLEMS.
c
c
            if(kbt.gt.1.or.icho.lt.5.or.k.gt.3)goto 1
              do 104 n=1,21,5
                sum=0.0
                sum1=0.0
                do 10 i=n,n+2
                  sum=cs(i)+sum
                  sum1=hi(i)+sum1
10              continue
                c(n)=sum/3.0
                h(n)=sum1/3.0
                i=n+3
                sum=0.0
                sum1=0.0
                do 11 j=i,i+1
                  sum=sum+cs(j)
                  sum1=hi(j)+sum1
11              continue
                c(i)=sum/2.0
104           h(i)=sum1/2.0
1           t=blkend
            jflag=0
            if(icho.ne.5) go to 6
            do 5 n=1,21,5
              do 3 i=n,n+2
                if(ibp(kbt).ne.i) goto 3
                  ibp(kbt)=n
3             continue
```

```
5       continue
c
c
c       NOW CALLING THE COST SUBROUTINE FOR THE
        BOTTLENECK(S) ITEMS FOR ONE-END ITEM PROBLEM(S)
c
c
6       if(icho.eq.1.and.ibp(kbt).eq.k)then
          call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
     1,blkstart,blkend,sumh,c,h,lk,name1)
          jflag=1
          kbt=kbt+1
        elseif
c
c
c       NOW CALLING THE COST SUBROUTINE FOR THE
c       BOTTLENECK(S) ITEMS FOR THREE-END ITEM PROBLEM(S)
c
c
     1(icho.eq.3.and.ibp(kbt).eq.k.or.icho.eq.3.and.
     2ibp(kbt)+1.eq.k.or.ibp(kbt)+2.eq.k.and.icho.
     3eq.3)then
c
c
          call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
     1,blkstart,blkend,sumh,c,h,lk,name1)
          jflag=1
          if(ich.eq.3)kbt=kbt+1
        elseif
c
c
c       NOW CALLING THE COST SUBROUTINE FOR THE
c       BOTTLENECK(S) ITEMS FOR FIVE-END ITEM PROBLEM(S).
c
c
     1(icho.eq.5.and.ibp(kbt).eq.k.or.ibp(kbt).eq.k-3.
     2and.icho.eq.5)then
          call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
     1,blkstart,blkend,sumh,c,h,lk,name1)
c
c
c       NOW STARTING THE DEMAND CALCULATION FOR FIVE-END
c       ITEM PROBLEMS.
c
c
          if(ich.eq.4)kbt=kbt+1
          jflag=1
        endif
        if(jflag.eq.1)go to 19
          do 91 j=blkstart,blkend
            sum=0.0
            dema(1,j)=0.0
            dema(4,j)=0.0
            do 92 i=1,3
              sum=sum+d(i,j)
```

```
92          continue
            dema(1,j)=sum+dema(1,j)
            sum=0.0
            do 93 i=4,5
              sum=sum+d(i,j)
93          continue
            dema(4,j)=sum+dema(4,j)
91          continue
            if(icho.eq.5)then
              eoq=int(sqrt(2.0*aver(ich,2)*c(k)/h(k))+0.5)
            else
              eoq=int(sqrt(2.0*aver(ich,2)*cs(k)/hi(k))+0.5)
            endif
            intvl=aver(ich,1)/eoq+0.999
            ord=int(aver(ich,1)/intvl)
c
c
c           ALL DEMANDS ARE PRODUCED IN PERIOD 1 TO CALCULATE
c           THE COST2
c
c
            idif1=0
            ico=0
            aver1=aver(ich,1)
            if(icho.eq.5.and.blkstart.eq.1)then
              stock1(lk,ich,blkstart-1)=0.0
              elseif(icho.ne.5.and.blkstart.eq.1)then
                stock1(k,ich,blkstart-1)=0.0
              elseif(icho.eq.5.and.blkstart.gt.1)then
                idif1=idif1+stock1(lk,ich,blkstart-1)
              else
                idif1=idif1+stock1(k,ich,blkstart-1)
c                     print*,'stock1(',k,ich,blkstart-
c       1,')=',stock1(k,ich,blkstart-1)
              endif
              do 80 j=blkstart,blkend
                if(icho.eq.5)then
                  if(stock1(lk,ich,j-1).ge.dema(ich,j))go to 81
                    ico=ico+1
                    itl(ico)=j
                    if((aver1-idif1).lt.aver(ich,1))go to 400
                      stock1(lk,ich,j)=aver(ich,1)+stock1(lk
          1,ich,j-1)-dema(ich,j)
                      idif1=aver1-stock1(lk,ich,j-1)+idif1
                    else
                    if(stock1(k,ich,j-1).ge.d(ich,j))go to 81
                      ico=ico+1
                      itl(ico)=j
                      if((aver1-idif1).lt.aver(ich,1))go to 400
                        stock1(k,ich,j)=aver(ich,1)+stock1(k
          1,ich,j-1)-d(ich,j)
                        idif1=aver1-stock1(k,ich,j-1)+idif1
                    endif
                    go to 80
400                 if(icho.eq.5)then
```

```
                idif1=aver1-idif1
                stock1(lk,ich,j)=idif1-dema(ich,j)
1+stock1(lk,ich,j-1)
              else
                idif1=aver1-idif1
                stock1(k,ich,j)=idif1-d(ich,j)
1+stock1(k,ich,j-1)
              endif
              go to 80
81            if(icho.eq.5)then
                stock1(lk,ich,j)=stock1(lk,ich,j-1)-
     1dema(ich,j)
              else
                stock1(k,ich,j)=stock1(k,ich,j-1)-
     1d(ich,j)
              endif
80     continue
       sum=0.0
       do 83 i=1,ico
         sum=sum+idif1*(t-itl(i)+1)
83     continue
       if(icho.eq.5)then
         cost2=c(k)*ico+h(k)*sum
       else
         cost2=cs(k)*ico+hi(k)*sum
       endif
       if(aver(ich,1).lt.eoq)go to 90
         idif1=0
         aver1=aver(ich,1)
         ico=0
         if(icho.eq.5.and.blkstart.eq.1)then
           stock2(lk,ich,blkstart-1)=0.0
         elseif(icho.ne.5.and.blkstart.eq.1)then
           stock2(k,ich,blkstart-1)=0.0
         elseif(icho.eq.5.and.blkstart.gt.1)then
           idif1=idif1+stock2(lk,ich,blkstart-1)
         else
           idif1=idif1+stock2(k,ich,blkstart-1)
         endif
         do 40 j=blkstart,blkend
           if(icho.eq.5)then
             if(stock2(lk,ich,j-1).ge.dema(ich,j))
1go to 20
           else
             if(stock2(k,ich,j-1).ge.d(ich,j))go to 20
           endif
           ico=ico+1
           itl(ico)=j
           if(ico.gt.intvl-1)go to 21
             if((aver1-idif1).lt.eoq)go to 21
               if(icho.eq.5)then
                 stock2(lk,ich,j)=stock2(lk,ich,j-1)+eoq-
1dema(ich,j)
               else
```

```
                stock2(k,ich,j)=stock2(k,ich,j-1)+eoq-
     1d(ich,j)
              endif
              idif1=idif1+eoq
              go to 40
21          ordl=aver(ich,1)-idif1
            if(icho.eq.5)then
              stock2(lk,ich,j)=stock2(lk,ich,j-1)+ordl-
     1dema(ich,j)
            else
              stock2(k,ich,j)=stock2(k,ich,j-1)+ordl-
     1d(ich,j)
            endif
            idif1=idif1+ordl
            go to 40
20          if(icho.eq.5)then
              stock2(lk,ich,j)=stock2(lk,ich,j-1)-
     1dema(ich,j)
            else
              stock2(k,ich,j)=stock2(k,ich,j-1)-d(ich,j)
            endif
40        continue
          sum=0.0
          if(ico.eq.1)go to 24
            do 23 j=1,ico-1
              sum=eoq*(t-itl(j)+1)+sum
23          continue
              sum=sum+ordl*(t-itl(ico)+1)
c
c
c       EOQ IS USED TO FIND THE COST3
c
c
              if(icho.eq.5)then
                cost3=c(k)*ico+h(k)*sum
              else
                cost3=cs(k)*ico+hi(k)*sum
              endif
              go to 25
24          sum=0.0
            do 50 j=1,ico
              sum=eoq*(t-itl(j)+1)+sum
50          continue
            if(icho.eq.5)then
              cost3=c(k)*ico+h(k)*sum
            else
              cost3=cs(k)*ico+hi(k)*sum
            endif
25          ord=int(ord)
            np1=aver(ich,1)-intvl*ord
            np0=intvl-np1
            iflag=0
            idif1=0
            aver1=aver(ich,1)
            ico=0
```

```
      ord2=0
      if(icho.eq.5.and.blkstart.eq.1)then
        stock3(lk,ich,blkstart-1)=0.0
      elseif(icho.ne.5.and.blkstart.eq.1)then
        stock3(k,ich,blkstart-1)=0.0
      elseif(icho.eq.5.and.blkstart.gt.1)then
        idif1=idif1+stock3(lk,ich,blkstart-1)
      else
        idif1=idif1+stock3(k,ich,blkstart-1)
      endif
      do 26 j=blkstart,blkend
        if(icho.eq.5)then
          if(stock3(lk,ich,j-1).ge.dema(ich,j))
     1go to 27
        else
          if(stock3(k,ich,j-1).ge.d(ich,j)) go to 27
        endif
        ico=ico+1
        itl(ico)=j
        if(iflag.eq.1) go to 281
          if(ico.le.np0)go to 28
            iflag=1
281     if((aver1-idif1).le.ord)go to 258
          ord1=ord+1
          if(icho.eq.5)then
            stock3(lk,ich,j)=ord1+stock3(lk,ich,j-1)-
     1dema(ich,j)
          else
            stock3(k,ich,j)=ord1+stock3(k,ich,j-1)-
     1d(ich,j)
          endif
          idif1=idif1+ord1
          go to 26
28      if((aver1-idif1).le.ord)go to 258
          if(icho.eq.5)then
            stock3(lk,ich,j)=ord+stock3(lk,ich,j-1)-
     1dema(ich,j)
          else
            stock3(k,ich,j)=ord+stock3(k,ich,j-1)-
     1d(ich,j)
          endif
          idif1=idif1+ord
          go to 26
258       ord2=aver1-idif1
          if(icho.eq.5)then
            stock3(lk,ich,j)=ord2-dema(ich,j)
     1+stock3(lk,ich,j-1)
          else
            stock3(k,ich,j)=ord2-d(ich,j)
     1+stock3(k,ich,j-1)
          endif
          idif1=idif1+ord2
          go to 26
27        if(icho.eq.5)then
```

```
                stock3(lk,ich,j)=stock3(lk,ich,j-1)-
     1dema(ich,j)
              else
                stock3(k,ich,j)=stock3(k,ich,j-1)-d(ich,j)
              endif
26          continue
            sum=0.0
            if(ico.eq.1)go to 62
              if(ord2.gt.0) go to 67
                do 29 i=1,np0
                  sum=sum+ord*(t-itl(i)+1)
29              continue
                if(np1.lt.1) go to 31
                  do 30 i=np0+1,np1+np0
                    sum=ord1*(t-itl(i)+1)+sum
30                continue
c
c
c           ORDL IS USED TO FIND THE COST4
c
c
31              if(icho.eq.5)then
                  cost4=c(k)*(np1+np0)+h(k)*sum
                else
                  cost4=cs(k)*(np1+np0)+hi(k)*sum
                endif
                go to 173
67           sum=0.0
             do 68 i=1,ico-1
               sum=sum+ord*(t-itl(i)+1)
68           continue
             sum=sum+ord2*(t-itl(ico)+1)
             if(icho.eq.5)then
               cost4=c(k)*ico+h(k)*sum
             else
               cost4=cs(k)*ico+hi(k)*sum
             endif
             go to 173
62        lflag=0
           do 63 i=1,ico
             sum=sum+ord*(t-itl(i)+1)
63         continue
           if(icho.eq.5)then
             cost4=c(k)+h(k)*sum
           else
             cost4=cs(k)+hi(K)*sum
           endif
           go to 173
90         sflag=0.0
           if(name1.ne.'roll')go to 118
             do 95 i=blkstart,blkend
               if(icho.eq.5)then
                 stock2(lk,ich,i)=stock1(lk,ich,i)
                 stock3(lk,ich,i)=stock1(lk,ich,i)
                 cost3=cost2
```

```
                cost4=cost2
              else
                stock2(k,ich,i)=stock1(k,ich,i)
                stock3(k,ich,i)=stock1(k,ich,i)
                cost3=cost2
                cost4=cost2
              endif
95          continue
            go to 173
118       cost3=cost2
          cost4=cost2
173       sumY=min(cost2,cost3,cost4)
          if(name1.ne.'roll')go to 7
c
c
c         NOW STARTING THE COST CALCULATION OF THE THREE
c         PERIOD FOR FUTURE USAGE FOR ROLLING SCHEDULE
c
c
          if(cost3.lt.cost2.or.cost4.lt.cost2) go to 51
            if(blkstart.eq.1)then
              sumhold=0.0
            else
              sumhold=suml(k)
            endif
            sumc=sumy+sumhold
            ico=0
            idif1=0
            aver1=aver(ich,1)
            if(icho.eq.5.and.blkstart.eq.1)then
              stock1(lk,ich,blkstart-1)=0.0
            elseif(icho.ne.5.and.blkstart.eq.1)then
              stock1(k,ich,blkstart-1)=0.0
            elseif(icho.eq.5.and.blkstart.gt.1)then
              idif1=idif1+stock1(lk,ich,blkstart-1)
            else
              idif1=idif1+stock1(k,ich,blkstart-1)
            endif
            do 35 j=blkstart,blkstart+2
              if(icho.eq.5)then
                if(stock1(lk,ich,j-1).ge.dema(ich,j))
     1go to 36
                  ico=ico+1
                  itl(ico)=j
                  if((aver1-idif1).lt.aver(ich,1))go to 263
                    stock1(lk,ich,j)=aver1-idif1
     1+stock1(lk,ich,j-1)-dema(ich,j)
                    idif1=aver1-stock1(lk,ich,j-1)+idif1
                  else
                    if(stock1(k,ich,j-1).ge.d(ich,j))
     1go to 36
                      ico=ico+1
                      itl(ico)=j
                      if((aver1-idif1).lt.aver(ich,1))
     1go to 263
```

```
                    stock1(k,ich,j)=aver1-idif1
     1+stock1(k,ich,j-1)-d(ich,j)
                    idif1=aver1-stock1(k,ich,j-1)+idif1
                endif
                go to 35
263             if(icho.eq.5)then
                  stock1(1k,ich,j)=aver1-idif1-dema(ich,j)
     1+stock1(1k,ich,j-1)
                else
                  stock1(k,ich,j)=aver1-idif1-d(ich,j)
     1+stock1(k,ich,j-1)
                endif
                idif1=aver1-idif1
                go to 35
36                if(icho.eq.5)then
                    stock1(1k,ich,j)=stock1(1k,ich,j-1)-
     1dema(ich,j)
                  else
                    stock1(k,ich,j)=stock1(k,ich,j-1)-
     1d(ich,j)
                  endif
35        continue
          if(ico.lt.1)go to 45
            do 37 i=1,ico
              if(icho.eq.5)then
                suml(k)=c(k)*ico+h(k)*(t-itl(ico)+1)*IDIF1
                sumn(k)=suml(k)
                sums(k)=suml(k)
              else
                suml(k)=cs(k)*ico+hi(k)*(t-itl(ico)+1)*IDIF1
                sumn(k)=suml(k)
                sums(k)=suml(k)
              endif
37          continue
            go to 7
45        suml(k)=0.0
          sumn(k)=0.0
          sums(k)=0.0
          go to 7
51        if(cost3.gt.cost4) go to 8
            if(blkstart.eq.1)then
              sumhold=0.0
            else
              sumhold=sumn(k)
          endif
          sumc=sumy+sumhold
          ico=0
          idif1=0
          aver1=aver(ich,1)
          if(icho.eq.5.and.blkstart.eq.1)then
            stock2(1k,ich,blkstart-1)=0.0
          elseif(icho.ne.5.and.blkstart.eq.1)then
            stock2(k,ich,blkstart-1)=0.0
            elseif(icho.eq.5.and.blkstart.gt.1)then
            idif1=idif1+stock2(1k,ich,blkstart-1)
```

```
      else
        idif1=idif1+stock2(k,ich,blkstart-1)
      endif
      do 9 j=blkstart,blkstart+2
        if(icho.eq.5)then
          if(stock2(lk,ich,j-1).ge.dema(ich,j))
     1go to 56
        else
          if(stock2(k,ich,j-1).ge.d(ich,j)) go to 56
        endif
        ico=ico+1
        itl(ico)=j
        if(ico.gt.(intvl-1)) go to 55
          if((aver1-idif1).lt.eoq)go to 55
            if(icho.eq.5)then
              stock2(lk,ich,j)=stock2(lk,ich,j-1)+eoq-
     1dema(ich,j)
            else
              stock2(k,ich,j)=stock2(k,ich,j-1)+eoq-
     1d(ich,j)
            endif
            idif1=idif1+eoq
            go to 9
55      ordl=aver(ich,1)-IDIF1
        if(icho.eq.5)then
          stock2(lk,ich,j)=stock2(lk,ich,j-1)+ordl-
     1dema(ich,j)
        else
          stock2(k,ich,j)=stock2(k,ich,j-1)+ordl-
     1d(ich,j)
        endif
        idif1=idif1+ordl
        go to 9
56        if(icho.eq.5)then
            stock2(lk,ich,j)=stock2(lk,ich,j-1)-
     1dema(ich,j)
          else
            stock2(k,ich,j)=stock2(k,ich,j-1)-d(ich,j)
          endif
9     continue
      suma=0.0
      if(ico.lt.1)go to 47
        do 13 j=1,ico
          suma=eoq*(t-itl(j)+1)+suma
13      continue
      if(icho.eq.5)then
        sumn(k)=c(k)*ico+h(k)*suma
        suml(k)=sumn(k)
        sums(k)=sumn(k)
      else
        sumn(k)=cs(k)*ico+hi(k)*suma
        suml(k)=sumn(k)
        sums(k)=sumn(k)
      endif
      go to 7
```

```
47          suml(k)=0.0
            sumn(k)=0.0
            sums(k)=0.0
            go to 7
8           if(cost4.gt.cost3.or.cost4.gt.cost2) go to 7
              if(blkstart.eq.1)then
                sumhold=0.0
              else
                sumhold=sums(k)
              endif
              sumc=sumy+sumhold
              ico=0
              ord=int(ord)
              np1=aver(ich,1)-intvl*ord
              np0=intvl-np1
              idif1=0
              aver1=aver(ich,1)
              iflag=0
              if(icho.eq.5.and.blkstart.eq.1)then
                stock3(lk,ich,blkstart-1)=0.0
              elseif(icho.ne.5.and.blkstart.eq.1)then
                stock3(k,ich,blkstart-1)=0.0
              elseif(icho.eq.5.and.blkstart.gt.1)then
                idif1=idif1+stock3(lk,ich,blkstart-1)
              else
                idif1=idif1+stock3(k,ich,blkstart-1)
              endif
              do 133 j=blkstart,blkstart+2
                if(icho.eq.5)then
                  if(stock3(lk,ich,j-1).ge.dema(ich,j))
        1go to 134
                else
                  if(stock3(k,ich,j-1).ge.d(ich,j))
        1go to 134
                endif
                ico=ico+1
                if(iflag.eq.1)go to 282
                  if(ico.le.np0)go to 135
                    iflag=1
282                 if((aver1-idif1).lt.ord)go to 267
                      ord1=ord+1
                      itl(ico)=j
                      if(icho.eq.5)then
                        stock3(lk,ich,j)=ord1+stock3(lk
        1,ich,j-1)-dema(ich,j)
                      else
                        stock3(k,ich,j)=ord1+stock3(k,ich,j-
        11)-d(ich,j)
                      endif
                      idif1=idif1+ord1
                      go to 133
135               if((aver1-idif1).lt.ord)go to 267
                    if(icho.eq.5)then
                      stock3(lk,ich,j)=ord+stock3(lk,ich,j-
        11)-dema(ich,j)
```

```
                  else
                    stock3(k,ich,j)=ord+stock3(k,ich,j-1)-
     1d(ich,j)
                  endif
                  itl(ico)=j
                  idif1=idif1+ord
                  go to 133
267               ord2=aver1-idif1
                  if(icho.eq.5)then
                    stock3(lk,ich,j)=ord2-dema(ich,j)
     1+stock3(lk,ich,j-1)
                  else
                    stock3(k,ich,j)=ord2-d(ich,j)
     1+stock3(k,ich,j-1)
                  endif
                  idif1=idif1+ord2
                  go to 133
134               if(icho.eq.5)then
                    stock3(lk,ich,j)=stock3(lk,ich,j-1)-
     1dema(ich,j)
                  else
                    stock3(k,ich,j)=stock3(k,ich,j-1)-
     1d(ich,j)
                  endif
133    continue
       sum=0.0
       if(ico.lt.1)go to 138
         if(ico.eq.1)go to 137
           do 139 i=1,ico-1
             sum=sum+ord*(t-itl(i)+1)
139        continue
           if(ico.le.np0)go to 141
             if(ord2.gt.0)then
               sum=sum+ord2*(t-itl(ico)+1)
             else
               sum=sum+ord1*(t-itl(ico)+1)
             endif
             go to 142
141        sum=sum+ord*(t-itl(ico)+1)
142        if(icho.eq.5)then
             sums(k)=c(k)*ico+h(k)*sum
             suml(k)=sums(k)
             sumn(k)=sums(k)
           else
             sums(k)=cs(k)*ico+hi(k)*sum
             suml(k)=sums(k)
             sumn(k)=sums(k)
           endif
           go to 7
137      if(icho.eq.5)then
           sums(k)=c(k)*ico+h(k)*ord*(t-itl(ico)+1)
           suml(k)=sums(k)
           sumn(k)=sums(k)
         else
           sums(k)=cs(k)*ico+hi(k)*ord*(t-itl(ico)+1)
```

```
            suml(k)=sums(k)
            sumn(k)=sums(k)
          endif
          go to 7
138       suml(k)=0.0
          sumn(k)=0.0
          sums(k)=0.0
7         if(name1.ne.'roll')go to 106
          do 111 i=blkstart,blkstart+2
            if(cost3.lt.cost2.or.cost4.lt.cost2)go to 48
              if(icho.eq.5)then
                stock2(lk,ich,i)=stock1(lk,ich,i)
                stock3(lk,ich,i)=stock1(lk,ich,i)
              else
                stock2(k,ich,i)=stock1(k,ich,i)
                stock3(k,ich,i)=stock1(k,ich,i)
              endif
              go to 111
48            if(cost3.gt.cost4.or.cost3.gt.cost2)go to 49
              if(icho.eq.5)then
                stock1(lk,ich,i)=stock2(lk,ich,i)
                stock3(lk,ich,i)=stock2(lk,ich,i)
              else
                stock1(k,ich,i)=stock2(k,ich,i)
                stock3(k,ich,i)=stock2(k,ich,i)
              endif
              go to 111
49            if(cost4.gt.cost3.or.cost4.gt.cost2)go to 111
              if(icho.eq.5)then
                stock1(lk,ich,i)=stock3(lk,ich,i)
                stock2(lk,ich,i)=stock3(lk,ich,i)
              else
                stock1(k,ich,i)=stock3(k,ich,i)
                stock2(k,ich,i)=stock3(k,ich,i)
              endif
111       continue
          go to 77
106       yflag=0.0
          do 107 i=blkstart,blkstart+2
            if(icho.eq.5)then
              stock1(lk,ich,i)=0.0
              stock2(lk,ich,i)=0.0
              stock3(lk,ich,i)=0.0
            else
              stock1(k,ich,i)=0.0
              stock2(k,ich,i)=0.0
              stock3(k,ich,i)=0.0
            endif
107       continue
          suml(k)=0.0
          sumn(k)=0.0
          sums(k)=0.0
          sumc=sumy
77        write(9,101)k,cost2,cost3,cost4,sumy,suml(k)
     1,sumn(k),sums(k)
```

```
101       format(i4,8f8.1)
19        return
          end
c
c
c         START OF SUBROUTINE COSTS
c         FUNCTIONS: TO CALCULATE THE COSTS FOR NON-
c         BOTTLENECK ITEM(S) USING SILVER MEAL TECNIQUE
c         PARAMETERS: KBT,K,PROD,STOCK,AVER,D,HI,CS,IBOTTLE,
c         IBP,SUMC,T,ICHO,BLKSTART,BLKEND,BLKSIZE,SUMH,SUML,
c         SUMN,SUMS,DEM,C,H,DEMA,LK,STOCK1,STOCK2,STOCK3,
c         SCOST,PROD1,STOCK4,PROD2,SUMSIL
c
c
          subroutine costs(kbt,k,prod,stock,aver,d,hi,cs
     1,ich,ibottle,ibp,sumc,t,icho,blkstart,blkend
     2,blksize,sumh,dem,c,h,dema,lk,stock1,stock2,stock3,
     3scost,prod1,stock4,prod2,sumsil,name1)
          integer blkstart,blkend
          dimension prod(25,5,96),stock(25,5,96),aver(5,3)
     1,hi(25),cs(25),it1(96),d(5,96),ibp(25),sumh(25)
     2,dem(5,96),c(25),h(25),dema(5,96),stock1(25,5,96)
     3,stock2(25,5,96),stock3(25,5,96),scost(25,5,96)
     4,prod1(25,5,96),stock4(25,5,96),prod2(25,5,96)
     5,sumsil(25)
          if(kbt.gt.1.or.icho.lt.5.or.k.gt.3)go to45
            do 42 n=1,21,5
              sum=0.0
              sum1=0.0
              do 43 i=n,n+2
                sum=cs(i)+sum
                sum1=hi(i)+sum1
43            continue
              c(n)=sum/3.0
              h(n)=sum1/3.0
              i=n+3
              sum=0.0
              sum1=0.0
              do 61 j=i,i+1
                sum=sum+cs(j)
                sum1=sum1+hi(j)
61            continue
              c(i)=sum/2.0
              h(i)=sum1/2.0
42          continue
45          t=blkend
            jflag=0
            if(icho.ne.5)go to 6
              do 5 n=1,21,5
                do 3 i=n,n+2
                  if(ibp(kbt).ne.i)go to 3
                    ibp(kbt)=n
3               continue
5             continue
c
```

```
c
c       NOW CALLING THE SUBROUTINE BOTTLE FOR THE
c       BOTTLENECK(S) FOR ONE-END ITEM PROBLEM FOR
c       BOTTLENECK(S) ITEM(S)
c
c
6       if(icho.eq.1.and.ibp(kbt).eq.k)then
           call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
     1,blkstart,blkend,sumh,c,h,lk,name1)
           jflag=1
           kbt=kbt+1
         elseif
c
c
c       NOW CALLING THE SUBROUTINE BOTTLE FOR THE
c       BOTTLENECK(S) FOR THREE-END ITEM PROBLEM(S) FOR
c       BOTTLENECK(S) ITEM(S)
c
c
     1(icho.eq.3.and.ibp(kbt).eq.k.or.icho.eq.3.
     2and.ibp(kbt)+1.eq.k.or.ibp(kbt)+2.eq.k.and.icho
     3.eq.3)then
        call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
     1,blkstart,blkend,sumh,c,h,lk,name1)
        jflag=1
        if(ich.eq.3)kbt=kbt+1
       elseif
c
c
c       NOW CALLING THE SUBROUTINE BOTTLE FOR THE
c       BOTTLENECK(S) FOR FIVE-END ITEM PROBLEM(S) FOR
c       BOTTLENECK(S) ITEM(S)
c
c
     1(icho.eq.5.and.ibp(kbt).eq.k.or.ibp(kbt)
     2.eq.k-3.and.icho.eq.5)then
        call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
     1,blkstart,blkend,sumh,c,h,lk,name1)
        if(ich.eq.4)kbt=kbt+1
          jflag=1
        endif
c
c
c       NOW STARTING THE DEMAND CALCULATION FOR FIVE-END
c       ITEM PROBLEM
c
c
        if(jflag.eq.1)go to 19
          do 91 j=blkstart,blkend
            sum=0.0
            dema(1,j)=0.0
            dema(4,j)=0.0
            do 92 i=1,3
92          sum=sum+d(i,j)
            dema(1,j)=sum+dema(1,j)
```

```
                sum=0.0
                do 93 i=4,5
93              sum=sum+d(i,j)
91            dema(4,j)=sum+dema(4,j)
              ik=0
              ic=1
              sum=0.0
              it=0
              do 20 j=blkstart,blkend
                if(icho.eq.5)then
                  if(stock4(lk,ich,j-1).ge.dema(ich,j))
     1go to 24
                    it=it+1
                    ic=ic+1
                    sum=sum+(ic-1)*h(k)*dema(ich,j)
                    scost(lk,ich,it)=sum/ic
                    if(scost(lk,ich,it).gt.scost(lk,ich,it-1))
     1go to 30
                      prod1(lk,ich,j)=prod1(lk,ich,j-1)
     1+dema(ich,j)
                      prod2(lk,ich,blkstart+ik)=prod1(lk,ich
     1,j)
                    else
                      if(stock4(k,ich,j-1).ge.d(ich,j))
     1go to 24
                        it=it+1
                        ic=ic+1
                        sum=sum+(ic-1)*hi(k)*d(ich,j)
                        scost(k,ich,it)=sum/ic
                        if(scost(k,ich,it).gt.scost(k,ich
     1,it-1))go to 30
                          prod1(k,ich,j)=prod1(k,ich,j-1)
     1+d(ich,j)
                          prod2(k,ich,blkstart+ik)=prod1(k
     1,ich,j)
                      endif
                      go to 20
24                  if(icho.eq.5)then
                      stock4(lk,ich,j)=stock4(lk,ich,j-1)-
     1dema(ich,j)
                      it=it+1
                      ik=ik+1
                      ic=0
                      sum=c(k)
                      scost(lk,ich,it)=sum
                    else
                      stock4(k,ich,j)=stock4(k,ich,j-1)-d(ich,j)
                      it=it+1
                      ik=ik+1
                      ic=0
                      sum=cs(k)
                      scost(k,ich,it)=sum
                    endif
20              continue
30              it=blkstart+ik
```

```
ic=1
if(j.gt.blkend)go to 51
  if(j.ge.blkend)ik=0
    if(icho.eq.5)then
      prod1(lk,ich,j)=dema(ich,j)
      prod2(lk,ich,j)=prod1(lk,ich,j)
      sum=c(k)
      scost(lk,ich,it)=sum
    else
      prod1(k,ich,j)=d(ich,j)
      prod2(k,ich,j)=prod1(k,ich,j)
      sum=cs(k)
      scost(k,ich,it)=sum
    endif
    l=0
    n=j+1
    do 25 m=n,blkend
      if(icho.eq.5)then
        if(stock4(lk,ich,m-1).ge.dema(ich,m))
1go to 27
          it=it+1
          ic=ic+1
          sum=sum+(ic-1)*h(k)*dema(ich,m)
          scost(lk,ich,it)=sum/ic
            if(scost(lk,ich,it).gt.scost(lk
1,ich,it-1))go to 50
              l=l+1
              prod1(lk,ich,m-1)=prod1(lk,ich
1,m-1)+dema(ich,m)
              prod2(lk,ich,m-1)=prod1(lk,ich
1,m-1)
            else
              if(stock4(k,ich,m-1).ge.d(ich
1,m))goto 27
                it=it+1
                ic=ic+1
                sum=sum+(ic-1)*hi(k)*d(ich,m)
                scost(k,ich,it)=sum/ic
                  if(scost(k,ich,it).gt.
1scost(k,ich,it-1))goto 50
                    l=l+1
                    prod1(k,ich,m-1)=prod1(k
1,ich,m-1)+d(ich,m)
                    prod2(k,ich,m-1)=prod1(k
1,ich,m-1)
                endif
                go to 25
27            if(m.gt.blkend)go to 51
                if(icho.eq.5)then
                  stock4(lk,ich,m)=stock4(lk,ich,m-1)
1-dema(ich,m)
                  l=l+1
                  it=it+1
                  ic=1
                  sum=c(k)
```

```
                    scost(lk,ich,m)=sum
                  else
                    stock4(k,ich,m)=stock4(k,ich,m-1)-
     ld(ich,m)
                    l=l+1
                    it=it+1
                    ic=1
                    sum=cs(k)
                    scost(k,ich,m)=sum
                  endif
                    go to 25
50          if(m.gt.blkend)go to 51
              l=0
              it=blkstart+ik
              ic=1
              if(icho.eq.5)then
                prod1(lk,ich,m)=dema(ich,m)
                prod2(lk,ich,m)=prod1(lk,ich,m)
                sum=c(k)
                scost(lk,ich,it)=sum
              else
                prod1(k,ich,m)=d(ich,m)
                prod2(k,ich,m)=prod1(k,ich,m)
                sum=cs(k)
                scost(k,ich,it)=sum
              endif
25          continue
51          t=blkend
            sum=0.0
            ico=0
            ssil=0.0
            do 46 j=blkstart,blkend
              if(icho.eq.5)go to 47
                if(prod2(k,ich,j).gt.0.0)then
                  ico=ico+1
                  sum=sum+hi(k)*(t-j+1)*prod2(k,ich,j)
                  stock4(k,ich,j)=prod2(k,ich,j)+stock4(k
          1,ich,j-1)-d(ich,j)
                else
                  stock4(k,ich,j)=prod2(k,ich,j)+stock4(k
          1,ich,j-1)-d(ich,j)
52              endif
                go to 46
47            if(prod2(lk,ich,j).gt.0.0)then
                sum=sum+h(k)*(t-j+1)*prod2(lk,ich,j)
                ico=ico+1
                stock4(lk,ich,j)=prod2(lk,ich,j)+stock4(lk
          1,ich,j)-dema(ich,j)
              else
                stock4(lk,ich,j)=prod2(lk,ich,j)+stock4(lk
          1,ich,j-1)-dema(ich,j)
53            endif
46          continue
            if(icho.eq.5)then
              sumy=c(k)*ico+sum
```

```
      else
        sumy=cs(k)*ico+sum
      endif
      if(name1.ne.'roll')then
        sumsil(k)=0.0
        sumc=sumy
        go to 44
      endif
      if(blkstart.eq.1)then
        sumhold=0.0
      else
        sumhold=sumsil(k)
      endif
      sumc=sumy+sumhold
      ico1=0
      do 31 j=blkstart,blkstart+2
        if(icho.eq.5)then
          if(prod2(lk,ich,j).le.0.0)go to 32
            ico1=ico1+1
            ssil=ssil+h(k)*(t-j+1)*prod2(lk,ich,j)
        else
          if(prod2(k,ich,j).le.0.0)go to 32
            ico1=ico1+1
            ssil=ssil+hi(k)*(t-j+1)*prod2(k,ich,j)
32       endif
31    continue
      if(icho.eq.5)then
        sumsil(k)=ico1*c(k)+ssil
      else
        sumsil(k)=ico1*cs(k)+ssil
      endif
44    write(9,103)k,sumy,sumc,sumsil(k)
103   format(i4,5x,3f13.2)
19    return
      end
c
c
c     START OF SUBROUTINE BOTTLE
c     FUNCTIONS: TO CALCULATE THE COSTS FOR BOTTLENECK
c                ITEM(S)
c     PARAMETERS:KBT,K,PROD,HI,CS,IBP,SUMC,T,ICHO,SUMH,
c                C,H,LK,PERIODS
c
c
      subroutine bottle(kbt,k,prod,hi,cs,ibp,sumc,t
     1,icho,blkstart,blkend,sumh,c,h,lk,name1)
      integer blkstart,blkend
      dimension prod(25,5,96),hi(25),cs(25),ibp(25)
     1,sumh(25),c(25),h(25)
      ip=0
      if(icho.eq.3)then
        ik=k/icho
        nk=k-ik*icho
        if(nk.eq.0)nk=3
      elseif(icho.eq.1)then
```

```
      nk=1
   else
      ik=k/icho
      nk=(k-ik*icho)
      if(nk.eq.0)nk=4
   endif
   sum=0.0
   suma=0.0
      do 1 j=blkstart,blkend
        if(icho.eq.5) goto 21
          if(prod(lk,nk,j).gt.0.0)then
            ip=ip+1
            sum=sum+hi(k)*(t-j+1)*prod(lk,nk,j)
          endif
          go to 1
21        if(prod(lk,nk,j).gt.0.0)then
            ip=ip+1
            sum=sum+h(k)*(t-j+1)*prod(lk,nk,j)
          endif
1       continue
      if(icho.eq.5)then
        cost1=sum+ip*c(k)
      else
        cost1=sum+ip*cs(k)
      endif
      if(name1.ne.'roll')then
        sumh(k)=0.0
        sumc=cost1
        go to 6
      endif
      if(blkstart.eq.1)then
        sumhold=0.0
      else
        sumhold=sumh(k)
      endif
      sumc=cost1+sumhold
      ip=0
      do 20 j=blkstart,blkstart+2
        if(icho.eq.5)go to 22
          if(prod(lk,nk,j).gt.0.0)then
            ip=ip+1
            suma=suma+hi(k)*(t-j+1)*prod(lk,nk,j)
          endif
          go to 20
22        if(prod(lk,nk,j).gt.0.0)then
            ip=ip+1
            suma=suma+h(k)*(t-j+1)*prod(lk,nk,j)
          endif
20      continue
      if(icho.lt.5)then
        costa=suma+ip*cs(k)
      else
        costa=suma+ip*c(k)
      endif
      sumh(k)=costa
```

```
6       write(9,10)k,cost1,sumc,sumh(k)
10      format(' position of bottleneck =',i4,f13.1
     1,f8.0,f8.0)
        return
        end
```

APPENDIX D
NAG Subroutine Program

```
c       This program is to show the uniform distribution
c       real*8 x
        integer i
c       double precision g05caf
       external g05caf
c        external g05cbf
        external g05ccf
        write (6,120)
c        call g05cbf(0)
        call g05ccf(0)
        do 20 i=1,20
           x=g05caf(92)
           write(6,121)x
20      continue
120     format(4(1x/),31h g05caf example program
      1results/1x)
121     format(1x,f10.4)
        stop
        end
```

APPENDIX E
Reprint of Published Papers

INT. J. PROD. RES., 1992, VOL. 30, NO. 4, 787–798

A heuristic for multi-level lot-sizing problems with a bottleneck

B. TOKLU† and J. M. WILSON

A simple heuristic is proposed for multi-level lot-sizing problems where there is a bottleneck. Previous methods to solve this problem have formulated the problem as an integer programming problem and solved the problem using a Lagrangian relaxation embedded within the branch and bound procedure. In this paper we suggest that items to be produced can be grouped into two types and a simple but efficient heuristic can be used to determine the production quantities required. A program was developed to compute production levels and was found to require only a small fraction of the computer time required by the full integer programming approach, and to produce solutions of reasonable quality. The heuristic is simple to implement.

1. Introduction

Lot-sizing in MRP only becomes realistic when features such as capacity constraints and the fact that systems are multilevel can be incorporated into the model. Blackburn and Millen (1982) review and add to contributions made to this area. Their work provides for simultaneous lot-sizing and capacity requirements planning in an MRP framework. However, one of the most successful attempts to tackle the multi-level lot-sizing problem with a bottleneck constraint has been by Billington *et al.* (1986). This paper will propose a simple heuristic approach to solve the problem modelled by Billington *et al.* (1986), and show that if the items for production are categorized into

(*a*) end-items, constrained by the bottleneck, and
(*b*) non-end-items, unconstrained,

then two simple procedures can be used independently, one for each category of item, to determine the production levels of each item. The reason for this categorization into two groups will be explained in section 3. Solutions will be sub-optimal, but of adequate quality, and are easy to obtain. The method to be proposed requires only a fraction of the computation required for solution of the integer programming formulation of the lot-sizing problem. In addition, the heuristic is easy to implement and program when compared with the Lagrangian heuristic approach of Billington *et al.* (1986), and should require much less computer time and have more practical appeal in a realistic setting.

In the next section the model developed by Billington *et al.* (1986) will be presented and the heuristic approach will be developed in the third section.

Revision received June 1991.

† Business School, Loughborough University of Technology, Loughborough, Leicestershire LE11 3TU, UK.

0020–7543/92 $3·00

2. Integer programming formulation of problem

For the lot-sizing problem a bottleneck will be defined as follows: a bottleneck is a work centre which converts raw materials into finished goods through the use of resources in the manufacturing process. Therefore a machine with limited capacity, highly skilled or specialized workers, and task-specific machines or tools can all be seen to be bottlenecks under this definition. All the resources could be classified into a bottleneck. Setup cost and time will be very important for all work stations, especially the bottleneck facility, and capacity limitations, which can result from either bottleneck capacity being greater than demand in the planning horizon or demand being exceeded by capacity from time to time. A general product structure with a bottleneck facility is given in Fig. 1 (from Billington *et al.* 1986).

The general product structure can be split into a number of special cases: (1) assembly (no commonality), (2) serial (one item, multi-stage) (3) parallel (a collection of serial structures which has a bottleneck in one of the stages), (4) single stage multi-item. This paper will concentrate on the case of a parallel structure.

As will be seen from the numbering system in Fig. 1, no item has a higher number than any of its predecessors. It is an *a priori* assumption that items in the bottleneck facility do not have predecessors (although this assumption can be relaxed for the subsequent heuristic approach). It can be seen that batching demands on product setups can result in capacity problems, and also affects predecessor items since the batches are passed through as dependent demands. Because capacity utilization varies through time, costs may not be constant.

Assumptions

1. All lead times between stages are assumed to be zero.
2. Demand for the multiple end items are assumed known and at constant known rate per year.
3. There is no demand for the components at any intermediate stages.
4. Back orders are not allowed.
5. The number of units coming from bill of material required in the production of one unit at the immediate successor stage to the other stage is assumed to be equal to one.
6. The unit production costs are assumed constant and hence are ignored.
7. Production must occur in advance of that demand.

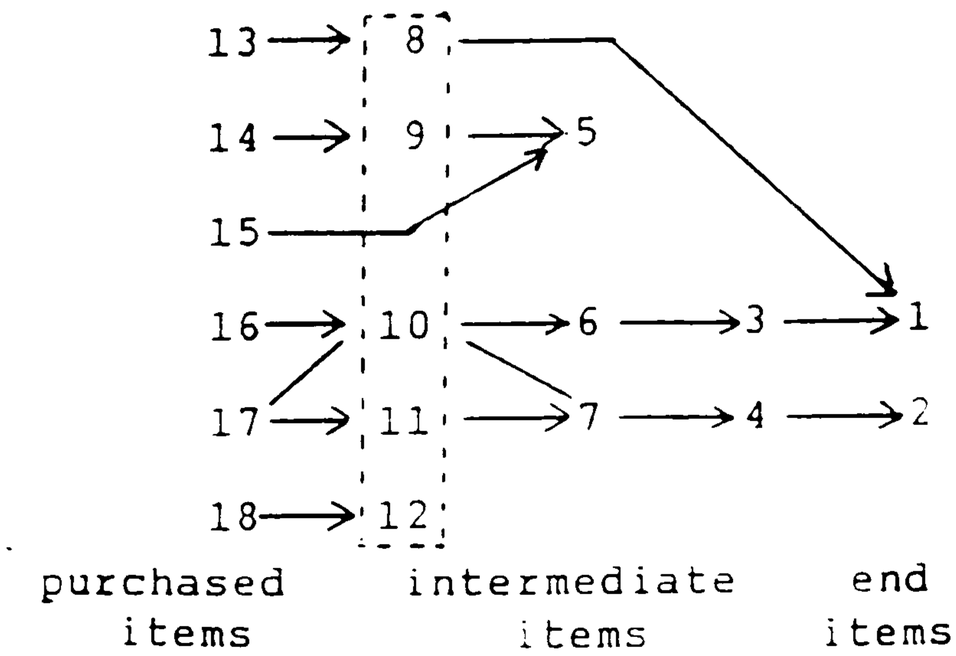


Figure 1. A general product structure with a bottleneck facility (the bottleneck facility is shown by the dashed lines).

Notation

a_{ij} the quantity of product i needed per unit production of product j; $a_{ij}=0$ for all $i<j$

b_i time needed on the bottleneck facility for the production of product i

cap_t available capacity of the work centre at time t

cs_i setup cost for product i (following the assumption made by Billington *et al.* (1986), for consistency, the possibility of 'carrying over' a setup from one period to another is not allowed)

d_{it} external (independent) demand for product i during time t

h_i holding cost for product i

I_{it} final inventory level of product i in period t

L_i lead time, which is the unavoidable time from the time the order is placed until it is available, for product i. This could be because of the time taken by a vendor to deliver a product, or could be a non-production lag

N number of products

P_{it} units of i produced in period t

s_i setup time for the work centre for product i. This takes the value zero for all items except those made on the work centre. This can also include processing time which is not related to the size of batch as in some heating operations

T total number of periods

X_{it} production indicator; equal to 1 if $P_{it}>0$ and zero otherwise

In the information below, inventory is eliminated by substituting cumulative production minus cumulative demand, and it is derived from a model in Billington *et al.* (1986).

Formulation

Minimize:

$$Z=\sum_{i=1}^{N}\sum_{t=1}^{T}[h_i(T-t+1)P_{it}+cs_iX_{it}] \tag{1}$$

Subject to:

$$\sum_{n=1}^{t}\left[P_{i,n-L_i}-\sum_{j=1}^{t}a_{ij}P_{jn}\right]\geqslant\sum_{n=1}^{t}d_{in}-I_{i0} \qquad i=1,\ldots,N \quad t=1,\ldots,T \tag{2}$$

$$\sum_{i=1}^{N}[b_iP_{it}+s_iX_{it}]\leqslant cap_t \qquad t=1,\ldots,T \tag{3}$$

$$X_{it}=\begin{cases}1 & \text{if } P_{it}>0\\ 0 & \text{otherwise}\end{cases} \qquad i=1,\ldots,N \tag{4}$$

$$P_{it}\geqslant 0 \qquad i=1,\ldots,N \quad t=1,\ldots,T \tag{5}$$

This is the formulation using integer programming when there is one capacity-constrained work centre.

In this formulation, constraint (2) illustrates that available production after subtracting the requirements is greater than or equal to external demand by eliminating the inventory in the planning horizon. Constraint (3) is the capacity constraint for the bottleneck facility, and (4) shows setup cost and time are applicable only if there is production.

In particular, Billington *et al.* (1986) concentrated on the three problem types 1-end item, 3-end items, and 5-end items and restricted these items so that they were the only ones affected by the bottleneck. In the computational testing N has maximum value 25 and T has maximum value 12, the values chosen in the work of Billington *et al.* (1986).

3. Simple heuristic for solution

The heuristic operates by first dividing production items into end-items and non-end-items. The reason for this is that production of each non-end item is unconstrained and so has neither any effect on the production of any other non-end-item nor on the production of the items which are constrained by the bottleneck. As demand required for all items is known in advance, the production decision for each non-end-item becomes a relatively simple one of when to produce in order to minimize the contribution to costs (from holding and setup costs) of each non-end-item. The fact that demand for end-items determines the demand for intermediate items does not invalidate the independence of production of each item as demand for end-items is known several periods in advance. The problem of determining when to produce end-items is more complex as these items must share the resources of the bottleneck. Thus for these items the production problem is a constrained problem. However, in general these items are in the minority.

Define S_{it} as stock of product i at start of period t, then

$$S_{it} = \sum_{n=1}^{t-1} P_{in} - d_{in}$$

3.1. *Non-end-items*

For these items an EOQ approach will be used. This approach was chosen as it is comparatively simple to operate and in general will produce solutions of good quality. Where demand levels are likely to be variable, an approach such as the Silver–Meal (1973) heuristic may be appropriate and is the subject for future investigation. Let Q_i be the EOQ for item i, based on setup cost cs_i and holding cost h_i. Then the following strategies are considered:

(*a*) Produce Q_i in period 1 and then next produce Q_i in the period when stocks would become negative if no production were made (i.e. find the next smallest t such that $S_{it} < d_{it}$).

Let t_i be the number of occasions on which item i will be produced. Then

$$t_i = \left[\sum_{t=1}^{T} d_{it}/Q_i + 0{\cdot}5 \right]$$

and production is made in any period n whenever $S_{in} < d_{in}$. Note that [ ] is the integer part function.

If in any period n

$$Q_i \geqslant \sum_{t=n}^{T} d_{it} - S_{in}$$

then set

$$Q_i = \sum_{t=n}^{T} d_{it} - S_{in}$$

(*b*) Let Z_i be the quantity of item i produced in period 1. The same quantity is next produced whenever stocks would become negative if no production were made (i.e. when $S_{in} < d_{in}$)

$$Z_i = \left(\sum_{t=1}^{T} d_{it} \right) \Big/ t_i$$

Continue this process through all the periods.

(*c*) Produce all items in period 1.

Strategies (*a*), (*b*) and (*c*) are evaluated to see which leads to the smaller total inventory cost over the N periods and then that strategy is chosen.

3.2. *End-items*

For these items a simple heuristic was adopted which would adopt a greedy approach to production by having few setups, but with heavy utilization of the resultant production capacity. In addition, the heuristic would operate in a cyclic manner, moving between items or sets of items in turn to produce reasonably smooth production. The approach has broad similarities with the work of McLain and Trigeiro (1985) except that by excluding setup time and cost they handle a problem that is easier to solve. Bahl and Ritzman (1984) also adopt a cyclic approach but do so by examining permutation schedules.

The heuristic will be described with reference to three cases.

Case (a). 1-end-item problems

Produce as much of end-item i as cap_t will allow in period 1, i.e. set $P_{i1} = cap_1$, then next produce i when stocks would become negative if no production were made, i.e. find the next smallest t for which $S_{it} < d_{it}$. Continue the process of producing in each period t which has this property.

$$\text{If } P_{in} \text{ would exceed } \sum_{t=n}^{T} d_{it} \text{ for any period } n$$

$$\text{then set } P_{in} = \sum_{t=n}^{T} d_{it}$$

Case (b). 3-end-item problems

A three period cycle is adopted.

Period 1 Set $P_{21} = d_{21}$, $P_{31} = d_{31}$ and $P_{11} = cap_1 - d_{21} - d_{31}$.

Period 2 Set $P_{32} = d_{32}$, $P_{12} = 0$ and $P_{22} = cap_2 - d_{32}$ provided $S_{12} > d_{12}$.
Otherwise set $P_{12} = d_{12} + d_{13} - S_{12}$, $P_{32} = d_{32}$ and
$P_{22} = cap_2 - P_{12} - P_{32}$.

Period 3 Set $P_{33} = cap_3$ and $P_{13} = P_{23} = 0$ provided $S_{13} > d_{13}$ and $S_{23} > d_{23}$.
Otherwise set $P_{13} = d_{13} - S_{13}$, $P_{23} = d_{23} + d_{24} - S_{23}$ and
$P_{33} = cap_3 - P_{13} - P_{23}$.

Period 4 Set $P_{14} = cap_4$ provided $S_{24} > d_{24}$ and $S_{34} > d_{34}$.

Period 5 Set $P_{25} = cap_5$ provided $S_{15} > d_{15}$ and $S_{35} > d_{35}$.

Period 6 Set $P_{36} = cap_6$ provided $S_{16} > d_{16}$ and $S_{26} > d_{26}$.

Again, if stocks of any product would become negative, produce sufficient of that product to satisfy demand over the next one or more periods until that product moves into the dominant production position.

Continue the process in the same cyclic manner for the remaining periods. If at any stage stocks of all products are sufficient for production to be zero in any period, no production is made in that period and the cycle for the appropriate product is delayed by one period.

Note. On the sample data on which the heuristic was tried, despite this data incorporating some highly variable demand levels and production capacities, it was found that none of the 'otherwise' type conditions listed above ever applied.

Case (c). 5-end-item problems

In order to keep the heuristic simple, more complex cases are now treated more in the style of case (b). The set of 5 products is simplified by considering products in just two sets rather than as 5 individual products. Here products $\{1, 2, 3\}$ are considered a set as are $\{4, 5\}$. This division was made purely for simplicity.

The two sets are now treated as single products and the cyclical approach of case (c), modified to a two-period cycle, is followed with the modification that when production of a product set can be larger than demand in that period, the production quantity is equal for each product in the set.

Total inventory cost is now the total of individual inventory costs arising from the end-items and the non-end-items.

4. Computational experience

The integer programming formulation solved by using the software MGG (1987) and SCICONIC (1986), and the heuristic approach of section 3, coded in Fortran, were compared on sets of data obtained from Billington (1983) and also discussed in Billington *et al.* (1986). In the experimental studies three different cost structures, three demand streams and three capacity levels were used. The three cost structures are detailed in Billington (1983). Essentially, for each structure the holding costs and the setup costs are set to different levels such that for structure 1 the average of the holding costs is 0·44 and the average for the setup costs is 400. For structures 2 and 3 the corresponding figures are 1·00, 340 and 0·64, 420 for holding cost and setup cost, respectively. The three demand streams are generated to give low (50%), medium (75%) and high (95%) demand. The results of the experiments are in Tables 1–9 and 27 problems were investigated. In each table, details of the linear programming (LP) solution to the integer programming (IP) formulation are given in the third column, details of the integer programming branch and bound approach are given in the fourth column, and of the heuristic solution in the right-hand column. The branching process by which the integer programming solutions were obtained was the standard default of the SCICONIC software which comprises an approach to choose subproblems which minimize the percentage error in the degradation of the objective function. The 'dynamic presolve' option of SCICONIC was also used, which aids branching exploration by tightening bounds where possible. The IP and heuristic solutions are compared to the LP optimum to give some indication of the quality of the solutions as the LP optimum provides a lower bound to the solution of the problem. In all cases the IP solution is not a proven optimal solution and the branch and bound process had to be cut off (before optimality could be proved) once a large amount of computer time had elapsed and further effort appeared unproductive. Computation was stopped after approximately 400 branch and bound nodes had been explored. The reason for this was that, although a number of problems were run 5000 or more branch and bound nodes, it was found that no better solution was obtained after that obtained in the first 100–300

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	10869	15511	13072
	Iterations	158	440	—
	CPU time	1·03	56·36	1·34
	Optimality*	—	0·70	0·83
75% utilization	Current value	10869	16610	13982
	Iterations	163	373	—
	CPU time	1·10	59·17	1·34
	Optimality	—	0·65	0·77
95% utilization	Current value	10899	17314	14827
	Iterations	178	260	—
	CPU time	1·19	69·24	1·34
	Optimality	—	0·63	0·74

* The percentage of LP solution.

Table 1. One-end-item problem with cost structure 1.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	30036	37085	35789
	Iterations	175	757	—
	CPU tume	1·15	52·86	3·32
	Optimality*	—	0·81	0·84
75% utilization	Current value	30036	38373	37373
	Iterations	176	1019	—
	CPU time	1·40	65·04	3·32
	Optimality	—	0·78	0·80
95% utilization	Current value	30031	39035	38720
	Iterations	171	572	—
	CPU time	1·20	39·68	3·32
	Optimality	—	0·77	0·74

* The percentage of LP solution.

Table 2. One-end-item problem with cost structure 2.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	18787	24777	23716
	Iterations	171	872	—
	CPU time	1·17	47·57	4·14
	Optimality*	—	0·76	0·79
75% utilization	Current value	18863	26770	24596
	Iterations	167	1182	—
	CPU time	1·18	58·24	4·14
	Optimality	—	0·70	0·77
95% utilization	Current value	19375	27684	25505
	Iterations	168	365	—
	CPU time	1·12	55·04	4·14
	Optimality	—	0·70	0·76

* The percentage of LP solution.

Table 3. One-end-item problem with cost structure 3.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	73623	91291	88097
	Iterations	519	1194	—
	CPU time	6·78	180·13	5·04
	Optimality*	—	0·81	0·84
75% utilization	Current value	73623	93900	87991
	Iterations	517	860	—
	CPU time	7·15	217·39	5·04
	Optimality	—	0·78	0·84
95% utilization	Current value	73951	95611	91925
	Iterations	511	869	—
	CPU time	7·29	206·22	5·04
	Optimality	—	0·77	0·80

* The percentage of LP solution.

Table 4. Three-end-item problem with cost structure 1.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	78962	98882	101095
	Iterations	507	711	—
	CPU time	6·89	143·84	6·11
	Optimality*	—	0·79	0·78
75% utilization	Current value	78976	99501	101217
	Iterations	480	1196	—
	CPU time	6·83	155·70	6·11
	Optimality	—	0·79	0·78
95% utilization	Current value	79772	102966	102877
	Iterations	507	1349	—
	CPU time	7·80	186·75	6·11
	Optimality	—	0·77	0·78

* The percentage of LP solution.

Table 5. Three-end-item problem with cost structure 2.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	58424	75418	72346
	Iterations	514	1136	—
	CPU time	6·61	181·64	6·54
	Optimality*	—	0·77	0·81
75% utilization	Current value	59061	72219	71670
	Iterations	487	1042	—
	CPU time	6·28	174·74	6·54
	Optimality	—	0·81	0·82
95% utilization	Current value	62103	78455	71815
	Iterations	497	924	—
	CPU time	6·49	183·98	6·54
	Optimality	—	0·79	0·86

* The percentage of LP solution.

Table 6. Three-end-item problem with cost structure 3.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	128948	159486	138185
	Iterations	997	1874	—
	CPU time	20·06	328·18	6·54
	Optimality*	—	0·81	0·93
75% utilization	Current value	128948	163464	137706
	Iterations	932	1420	—
	CPU time	19·92	396·08	6·54
	Optimality	—	0·79	0·94
95% utilization	Current value	129030	175342	137612
	Iterations	949	1805	—
	CPU time	20·61	356·86	6·54
	Optimality	—	0·74	0·94

* The percentage of LP solution.

Table 7. Five-end-item problem with cost structure 1.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	107807	131881	108674
	Iterations	952	1318	—
	CPU time	18·73	396·75	8·16
	Optimality*	—	0·82	0·99
75% utilization	Current value	107856	135130	109484
	Iterations	943	1709	—
	CPU time	19·30	397·21	8·16
	Optimality	—	0·80	0·99
95% utilization	Current value	108703	143170	112093
	Iterations	912	2098	—
	CPU time	19·29	416·76	8·16
	Optimality	—	0·76	0·97

* The percentage of LP solution.

Table 8. Five-end-item problem with cost structure 2.

		LP solution	IP solution	Heuristic solution
50% utilization	Current value	95601	120514	102001
	Iterations	1001	1340	—
	CPU time	19·75	388·49	8·56
	Optimality*	—	0·78	0·94
75% utilization	Current value	96159	122071	101352
	Iterations	970	2332	—
	CPU time	19·43	411·54	8·56
	Optimality	—	0·79	0·95
95% utilization	Current value	97624	127091	101440
	Iterations	1000	2219	—
	CPU time	22·95	526·00	8·56
	Optimality	—	0·70	0·96

* The percentage of LP solution.

Table 9. Five-end-item problem with cost structure 3.

nodes, so 400 nodes was taken as a convenient stopping point. CPU times quoted are for the time (in seconds) taken on a Hewlett Packard 9000 to reach the given solution. A number of features are evident from the results quoted in the tables.

(1) The heuristic approach is rapid, taking only a few seconds of CPU time.
(2) In all but three cases (those in Table 5) the heuristic solutions are better than the IP solutions.

For 5-end-item problems the heuristic seems to work particularly well, in the sense that the solutions are close to the lower bound value. However, this may only mean that the lower bounds are tighter for these problems. These tighter lower bounds may arise because more realistic values of the effect of setup costs (which are unrealistically reduced in the LP relaxation of the IP formulation) are likely to arise when there are more types of items being produced in the bottleneck and so more fractions of the true setup cost to add together.

For the three cases where the heuristic does not work well it may be that rather more setups than necessary are being used when setup cost is above average.

The IP solutions obtained were not optimal, but it should be noted that the work of Billington *et al.* (1986) was also unable to make comparison of its Lagrangean solutions with optimal solutions, except in a few restricted cases.

It is impossible to compare precisely the heuristic approach of this paper with that of Billington *et al.* (1986) as we do not have access to their program. It was felt that once rapid solution times for the heuristic approach were obtained (all less than 8·56 CPU seconds) and such high quality solutions were obtained for the 5-end-item problems that it was not appropriate to program the Lagrangean approach. What was of interest was to find a simpler approach. It is likely that their method produces good quality solutions but the code is complex and unlikely to operate as rapidly as the few seconds required here. In addition, the heuristic of this paper is essentially a 'back of an envelope' approach organized into a computer program, which should have appeal to the engineer responsible for scheduling because the approach works on a few simple principles which can still be used when the assumptions made in the original model are

relaxed to accommodate more realistic operating conditions. The approach is sufficiently flexible to provide quick solutions for a variety of extensions of the basic problem and the approach is appropriate for quick reworking of schedules whenever changes occur in demand streams or breakdowns occur.

5. Summary

A simple heuristic for the bottleneck multi-level lot-sizing problem has been developed. The heuristic provides quick and easy solutions for the problem and is sufficiently simple to be used even without a computer routine.

Acknowledgment

The authors would like to thank Professor P. J. Billington for making data available to us and for providing encouragement and advice. They are indebted to anonymous referees for helpful suggestions and pointing out some references of interest. In addition, they would like to thank Gazi University, Turkey for financial support given to the first author.

References

BAHL, H. C., and RITZMAN, L. P., 1984, A cyclical scheduling for lot sizing with capacity constraints. *International Journal of Production Research*, **22**, 791–800.

BILLINGTON, P. J., 1983, Multi-level lot-sizing with a bottleneck work center. Ph.D. Dissertation, Cornell University.

BILLINGTON, P. J., BLACKBURN, J. D., MAES, J., MILLEN, R. A., and VAN WASSENHOVE, L. W., 1988, Multi-product scheduling in multi-stage serial systems. In A. Chikan and M. C. Lovell (eds), *The Economics of Inventory Management* (Amsterdam: Elsevier Science).

BILLINGTON, P. J., MCLAIN, J. O., and THOMAS, L. J., 1986, Heuristics for multi-level lot-sizing with a bottleneck. *Management Science*, **32**, 989–1006.

BLACKBURN, J. D., and MILLEN, R. A., 1982, Improved heuristics for multi-stage requirements planning systems. *Management Science*, **28**, 44–56.

MCLAIN, J. O., and TRIGIERO, W. W., 1985, Cyclic assembly schedules. *IIE Transactions*, **17**, 346–353.

MGG USER GUIDE, 1987, SD-Scicon, Milton Keynes, England.

SCICONIC USER GUIDE, 1986, SD-Scicon, Milton Keynes, England.

SILVER, E. A., and MEAL, H. C., 1973, A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. *Production and Inventory Management*, **14**, 64–74.

A Heuristic for Multilevel Lot-Sizing Problems with Multiple Bottleneck

by

B. Toklu and J.M. Wilson

Paper 1991:14

LOUGHBOROUGH UNIVERSITY
MANAGEMENT RESEARCH SERIES

PAPER NUMBER 1991:14

Abstract

A simple heuristic is proposed for multilevel lot-sizing problems where there are multiple bottlenecks. Previous methods to solve this problem with one bottleneck have formulated the problem as an integer programming problem and solved the problem using a Lagrangian relaxation embedded within the branch and bound procedure. Other approaches have used simple heuristics.

In this paper we suggest that items to be produced can be grouped into two types and a simple but efficient heuristic can be used to determine the production quantities required. A program was developed to compute production levels and was found to require only a small fraction of the computer time required by the full integer programming approach and to produce solutions of reasonable quality. The heuristic is simple to implement.

Keywords: heuristics, inventory, lot-sizing

1. INTRODUCTION

Models for lot-size problems in MRP only become realistic when features such as capacity constraints and the fact that systems are multilevel can be incorporated. Blackburn and Millen (1982) reviews and adds to contributions made to this area. Their work provides for simultaneous lot-sizing and capacity requirements planning in an MRP framework. However, one of the most succesful attempts to tackle the multilevel lot-sizing problem with a bottleneck constraint has been by Toklu and Wilson (1991). This paper will extend their previous heuristic approach to solve the multiple bottleneck problem and show that if the items for production are categorised into

(a) items constrained by the bottleneck

(b) items which are unconstrained

then two simple procedures can be used independently, one for each category of item, to determine the production levels of each item. The reason for this categorisation into two groups will be explained in section 3. Solutions will be sub-optimal but of adequate quality and are easy to obtain. The method to be proposed requires only a fraction of the computation required for solution of the integer programming formulation of the lot-sizing

problem. In addition the heuristic is easy to implement and program when compared to to the Lagrangian heuristic approach used in Billington et al. (1986) and modified to handle another bottleneck facility. The approach of this paper should require much less computer time and have more practical appeal in a realistic setting.

In the next section the model developed by Billington et al. (1986) will be extended to include multiple bottlenecks and the heuristic approach will be developed in the third section.

2. INTEGER PROGRAMMING FORMULATION OF PROBLEM

For the lot-sizing problem a bottleneck will be defined as follows. A bottleneck is a work centre which converts raw materials into finished goods through the use of resources in the manufacturing process. Therefore a machine with limited capacity, highly skilled or specialised workers, and task-specific machines or tools can all be seen to be bottlenecks under this definition. All the resources could be classified into a bottleneck. Set up cost and time will be very important for all work stations, especially the bottleneck facility and capacity limitations, which can result from either bottleneck capacity being greater than demand in the planning horizon or demand being exceeded by capacity from time to

time. A general product structure with a bottleneck facility is given in Figure 1 (from Billington et al. (1986)).

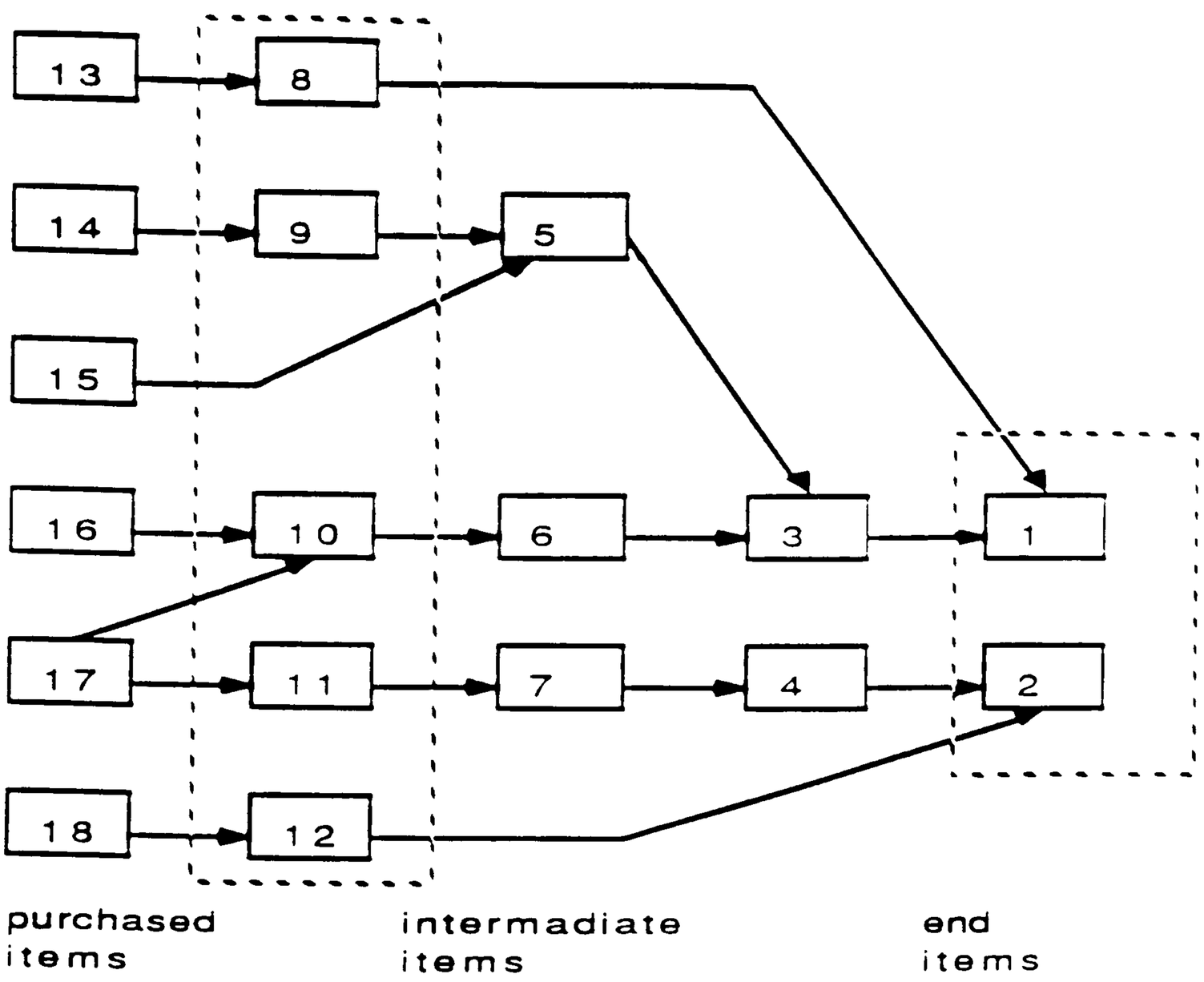


Figure 1. A General Product Structure with multiple Bottleneck Facility (The Bottleneck facilities are shown by the dashed lines)

The General product structure can be split into a number of special cases: (1) assembly (no commonality), (2) serial (one item, multi - stage) (3) parallel (a collection of serial structures which has a bottleneck in one of the stages), (4) single stage multi-item. This paper will concentrate on the case of a parallel structure with multiple bottlenecks.

As will be seen from the numbering system in Figure 1, no item has a higher number than any of its prodecessors. It is an a priori assumption that items in the bottleneck facility do not have predecessors (altough this assumption can be relaxed for the subsequent heuristic approach). It can be seen that batching demands on product setups can result in capacity problems and also affects predecessor items since the batches are passed through as dependent demands. Because capacity utilisation varies through time, costs may not be constant.

Assumptions

1. All lead times between stages are assumed to be zero,
2. Demand for the multiple end items are assumed known and at constant known rate per year,
3. There is no demand for the components at any intermediate stages,
4. Back orders are not allowed,
5. The number of units coming from bill of material required in the production of one unit at the immediate successor stage to the other stage is assumed to be equal to one,
6. The unit production costs are assumed constant and hence are ignored,

7. Production must occur in advance of that demand.

Notation

a_{ij} = The quantity of product i needed per unit production of product j; $a_{ij} = 0$ for all $i < j$,

b_{il} = Time needed on the bottleneck facility l for the production of product i,

cap_t = Available capacity of the work centre at time t,

cs_i = Setup cost for product i (following the assumption made by Billington et al. (1986), for consistency, the possibility of " carrying over " a setup from one period to another is not allowed),

d_{it} = External (independent) demand for product i during time t,

h_i = Holding cost for product i,

I_{it} = Final inventory level of product i in period t,

L_i = Lead time, which is the unavoidable time from the time the order placed until it is available, for product i. This could be because of the time taken by a vendor to deliver a product, or could be a non-production lag,

N = Number of products,

P_{it} = Units of i produced in period t,

S_{il} = Setup time for the work centre l for product i. This takes the value zero for all items except those made on the work centre. This can also

include processing time which is not related to the size of batch as in some heating operations,

T = Total number of periods,

M = The number of bottlenecks,

X_{it} = Production indicator; equal to 1 if $P_{it} > 0$ and zero otherwise.

In this information below inventory is eliminated by substituting cumulative production minus cumulative demand, and it is derived from a model in Billington et al. (1986).

Formulation

Minimise:

$$Z = \sum_{i=1}^{N} \sum_{t=1}^{T} [\, h_i \,(\, T-t+1 \,)\, P_{it} + cs_i X_{it} \qquad (1)$$

Subject to:

$$\sum_{n=1}^{t} [\, P_{i,n-1i} - \sum_{j=1}^{t} a_{ij} P_{jn} \,] \geq \sum_{n=1}^{t} d_{in} - I_{io} \qquad \begin{matrix} i=1,\ldots,N \\ t=1,\ldots,T \end{matrix} \quad (2)$$

$$\sum_{i=1}^{N} [\, b_{il} P_{it} + s_{il} X_{it} \,] \leq cap_t \qquad \begin{matrix} t=1,\ldots,T \\ l=1,\ldots,M \end{matrix} \quad (3)$$

$$X_{it} = \begin{cases} 1 & \text{if } P_{it} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad i=1, \quad ,N \quad (4)$$

$$P_{it} \geq 0 \quad i=1,\ldots,N \text{ and } t=1,\ldots,T \qquad (5)$$

This is the formulation using Integer Programming when there are multiple capacity constrained work centres.

In this formulation, constraint (2) illustrates that available production after subtracting the requirements is greater than or equal to the external demand by eliminating the inventory in the planning horizon. Constraint (3) is the capacity constraint for the bottleneck facilities, and (4) shows setup cost and time are applicable only if there is production.

In their previous paper Toklu and Wilson (1991) concentrated on the three problem types: 1-end item, 3-end items, and 5-end items and restricted these items so that they were the only ones affected by the single bottleneck. In the computational testing N had maximum value 25 and T had maximum value 12, the values chosen in the work of Billington et al. (1986)

3. SIMPLE HEURISTIC FOR MULTIPLE BOTTLENECK

All items which are produced by a manufacturing firm are assumed independent from each other. So the proposed heuristic operates by first dividing production items into end-items and non-end-items. The reason for this is that production of each non-end item is unconstrained and so has neither any effect on the production of any other non-end-item nor on the production of the items which are constrained by the bottleneck. In what follows it will be assumed that there are just two bottlenecks. The second bottleneck occurs in either non-end-items or end-items in

the product structure. In our case one bottleneck will always be taken to appear in the end-items and the second one could be any other intermediate or raw material stage. The program, written in Fortran 77, which will subsequently be discussed, was designed to handle cases irrespective of where the bottlenecks occur. As demand required of all items is known in advance, the production decision for each no-end-item becomes a relatively simple one of when to produce in order to minimise the contribution to costs (holding and setup costs) of each non-end-item.

Non-End-Items outside Bottleneck

The fact that demand for end-items determines the demand for intermediate items does not invalidate the independence of the production of each item as demand for end-items is known several periods in advance. The problem of determining when to produce end-items is more complex as these items must share the resources of the bottleneck. Thus for these items the production problem is a constrained problem. However, in general these items are in the minority.

Define S_{it} as stock of product i at start of period t

$$\text{then} \qquad S_{it} = \sum_{n=1}^{t-1} P_{in} - d_{in} \quad .$$

For these items an EOQ approach will be used. This approach was chosen as it is comparatively simple to operate and in general will produce solutions of good quality. Where demand levels are likely to be variable, an approach such as the Silver-Meal (1973) heuristic may be appropriate and is the subject for future investigation.

Let Q_i be the EOQ for item i, based on setup cost cs_i and holding cost h_i. Then the following strategies are considered:

(a) Produce Q_i in period 1 and then next produce Q_i in the period when stocks would become negative if no production were made (i.e. find the next smallest t such that $S_{it} < d_{it}$).

Let t_i be the number of occasions on which item i will be produced. Then

$$t_i = \left[\sum_{n=1}^{t-1} d_{it} \; / \; Q_i + 0.5 \right]$$

and production is made in any period n whenever $S_{in} < d_{in}$.

Note that [ ] is the integer part function.

If in any period n

$$Q_i \geq \sum_{t=n}^{T} d_{it} - S_{in} \quad \text{then set } Q_i = \sum_{t=n}^{T} d_{it} - S_{in} .$$

(b) Let Z_i be the quantity of item i produced in period 1. The same quantity is next produced whenever stocks would become negative if no production were made (ie when $S_{in} < d_{in}$).

$$Z_i = \left(\sum_{t=1}^{T} d_{it} \right) / t_i$$

Continue this process through all the periods.

(c) Produce all items in period 1.

Strategies (a), (b), and (c) are evaluated to see which leads to the smaller total inventory cost over the N periods and then that strategy is chosen.

Non-End-Items with Bottleneck

Assuming that the second bottleneck has been located in 3^{rd} item in 1-end-item problem. We have now got one bottleneck in the intermediate level in the 1-end-item problem for example. To deal with this kind of problem, we note that the non-bottleneck items have been explained for different kinds of end-item problems. Now the bottleneck items in the non-end items will be explained .

For these items a simple heuristic was adopted which would adopt a greedy approach to production by having few setups, but with heavy utilisation of the resultant production capacity. In addition, the heuristic would operate in a cylic manner moving between items or sets of items in turn to produce reasonably smooth production. The approach has broad similarities with the work of Mclain and Trigerio (1985) except that by excluding setup time and cost they handle a problem that is easier to solve. Bahl and Ritzman (1984) also adopt a cyclic approach but do so by examining permutation schedules.
The heuristic will be described with reference to three cases.

Case (a) 1-end-item problems

Assuming that one of the bottleneck has been located in the 3rd item in the product structure, the heuristic is that which poduces as much of non- end-item i as cap_t will allow in period 1 i.e. set $P_{i1} = cap_1$, then next produce i when stocks would become negative if no production were made, i.e. find the next smallest t for which $S_{it} < d_{it}$.

Continue the process of producing in each period t which has this property.

If P_{in} would exceed $\sum_{t=n}^{T} d_{it}$ for any period n

then set $P_{in} = \sum_{t=n}^{T} d_{it}$.

Case (b) 3-end-item problems

The bottleneck is located in the 7^{th}, say, item(s) in the product structure and according to the heuristic, the first priority is in the first item in the structure, then the second item and so on.

A three period cycle is adopted.

Period 1 Set $P_{21} = d_{21}$, $P_{31} = d_{31}$ and $P_{11} = cap_1 - d_{21}$ - d_{31} .

Period 2 Set $P_{32} = d_{32}$, $P_{12} = 0$ and $P_{22} = cap_2 - d_{32}$ provided $S_{12} > d_{12}$.

Otherwise set $P_{12} = d_{12} + d_{13} - S_{12}$, $P_{32} = d_{32}$ and $P_{22} = cap_2 - P_{12} - P_{32}$.

Period 3 Set $P_{33} = cap_3$ and $P_{13} = P_{23} = 0$ provided $S_{13} > d_{13}$ and $S_{23} > d_{23}$.

Otherwise set $P_{13} = d_{13} - S_{13}$, $P_{23} = d_{23} + d_{24} - S_{23}$ and $P_{33} = cap_3 - P_{13} - P_{23}$.

Period 4 Set $P_{14} = cap_4$ provided $S_{24} > d_{24}$ and $S_{34} > d_{34}$.

Period 5 Set $P_{25} = cap_5$ provided $S_{15} > d_{15}$ and $S_{35} > d_{35}$.

Period 6 Set $P_{36} = cap_6$ provided $S_{16} > d_{16}$ and $S_{26} > d_{26}$.

Again if stocks of any product would become negative produce sufficient of that product to satisfy demand over the next one or more periods until that product moves into the dominant production position.

Continue the process in the same cyclic manner for the remaining periods. If at any stage stocks of all products are sufficient for production to be zero in any period, no production is made in that period and the cycle for the appropriate product is delayed by one period.

Note: on the sample data on which the heuristic was tried despite this data incorporating some highly variable demand levels and production capacities, it was found that none of the " otherwise " type conditions listed above ever applied.

Case (c) 5-end-item problems

The bottleneck is located in the 11^{th} item(s), say, in the product structure. In order to keep the heuristic simple, more complex cases are now treated more in the style of Case (b). The set of 5 products is simplified by considering products in just two sets rather than as 5 individual products. Here products { 1,2,3 } are considered a set as are { 4,5 }. This division was made purely for simplicity.

The two sets are now treated as single products and the cyclical approach of case (c), modified to a two-period cycle, is followed with the modification that when production of a product set can be larger than demand in that period, the production quantity is equal for each product in the set.

Total inventory cost is now the total of individual inventory costs arising from the end-items and the non-end items.

4. End Items with Bottleneck

We have assumed that there is always a bottleneck in the end items. To solve this problem, the same rule which was used to explain non-end items problem with bottleneck is used with one difference : instead of item(s) 7^{th}, 1^{st} item(s) will be used in the three end-item problem, in the one end-item problem item 1 will be the bottleneck instead of item 3, and finally in the five end item problem item(s) 1^{st} will be used instead of 11^{th} item(s) which is grouped in the same way as previously.

5. COMPUTATIONAL EXPERIENCE

The integer programming formulation solved by using the software MGG (1987) and SCICONIC (1986), and the heuristic approach, coded in Fortran77, of Section 3 were

compared on sets of data obtained from Billington (1983) and also discussed in Billington et al. (1986) but modified to incorporate two bottlenecks, in the experimental studies three different cost structures, three demand streams and three capacity levels were used. The three cost structures are detailed in Billington (1983). Essentially for each structure the holding costs and the setup costs are set to different levels such that for structure 1 the average of the holding costs is 0.44 and the average for the setup costs is 400. For structures 2 and 3 the corresponding figures are 1.00, 340 and 0.64, 420 for holding cost and setup cost, respectively. The three demand streams are generated to give low (50%), medium (75%) and high (95%) demand. The results of the experiments are in Tables 1 - 3 and 9 problems were investigated. In each table, details of the linear programming (LP) solution to the integer programming (IP) formulation are given in the third column, details of the integer programming branch and bound approach are given in the fourth column, and of the heuristic solution in the right hand column. The branching process by which the integer programming solutions were obtained was the standard default of the SCICONIC software which comprises an approach to choose subproblems which minimise the percentage error in the degradation of the objective function. The " dynamic presolve " option of SCICONIC was also used which aids

branching exploration by tightening bounds where possible. The IP and heuristic solutions are compared to the LP optimum to give some indication of the quality of the solutions as the LP optimum provides a lower bound to the solution to the problem. In all cases the IP solution is not a proven optimal solution and the branch and bound process had to be cut off, before optimality could be proved, once a large amount of computer time elapsed and further effort appeared unproductive. Computation was stopped after approximately 400 branch and bound nodes had been explored. The reason for this was that although a number of problems were run 5000 or more branch and bound nodes, it was found that no better solution was obtained in the first 100 - 300 nodes, so 400 nodes was taken as a convenient stopping point. CPU time quoted are for the time (in seconds) taken on a Hewlett Packard 9000 to reach the given solution. A number of features are evident from the results quoted in the tables.

1 The heuristic approach is rapid taking a few seconds of CPU time

2. In all but one case (those in Table 2) the heuristic solutions are better than the IP solutions.

For 5 end-item problem the heuristic seems to work particularly well, in the sense that the solutions are

close to the lower bound value. However, this may only mean that the lower bounds are tighter for these problems. These tighter lower bounds may arise because more realistic values of the effect of setup costs (which are unrealistically reduced in the LP relaxation of the IP formulation) are likely to arise when there are more types of items being produced in the bottleneck and so more fractions of the true setup cost to add together. For the three cases where the heuristic does not work well it may be that rather more setups than necesary are being used when setup cost is above average.

It was felt that once rapid solution times for the heuristic approach were obtained (all less than 20.32 CPU seconds) and such high quality solutions were obtained for the 5 end-item problems that it was not appropriate to program the Lagrangean approach. What was of interest was to find a simpler approach. It is likely that their method produces good quality solutions but the code is complex and unlikely to operate as rapidly as the few seconds required here. In addition, the heuristic of this paper is essentially a "back of an envelope" approach organised into a computer program, which should have appeal to the engineer responsible for scheduling because the approach works on a few simple principles which can still be used when the assumptions made in the original model are relaxed to accomodate more realistic operating conditions. The approach is sufficiently

flexible to provide quick solutions for a variety of extensions of the basic problem and the approach is appropriate for quick reworking of schedules whenever changes occur in demand streams or breakdowns occur.

5. SUMMARY

A Simple heuristic for the multiple bottleneck multi - level lot-sizing problem has been developed. The heuristic provides quick and easy solutions for the problem and is sufficiently simple to be used even without a computer routine.

Acknowledgement

The authors would like to thank Professor P J Billington for making data available to us and for providing encouragement and advice. In addition, they would like to thank Gazi University, Turkey for financial support of the first author.

Table 1. One End Item Problem with Cost Structure 1

		LP Solution	IP Solution	Heuristic Solution
50% and 20% util.	Current Value	10869.0	15525.7	13082.0
	Iterations	152	388	-
	CPU Time	0.94	16.74	2.14
	Optimality*	-	0.70	0.83
75% and 50% util.	Current Value	18869.0	17274.8	14346.5
	Iterations	152	667	-
	CPU Time	0.92	23.52	2.14
	Optimality	-	0.62	0.75
95% and 75% util.	Current Value	10898.0	18822.2	15801.5
	Iterations	159	1013	-
	CPU Time	0.94	24.88	2.14
	Optimality	-	0.57	0.68

* The Percentage of LP Solution

Table 2. Three End Item Problem with Cost Structure 1

		LP Solution	IP Solution	Heuristic Solution
50% and 25% util.	Current Value	73623.4	92583.3	102003.9
	Iterations	424	764	-
	CPU Time	5.36	58.44	13.23
	Optimality	-	0.79	0.72
75% and 50% util.	Current Value	74066.4	92695.1	92639.4
	Iterations	453	1224	-
	CPU Time	6.04	63.74	13.23
	Optimality	-	0.79	0.79
95% and 75% util.	Current Value	73902.2	97701.0	94046.9
	Iterations	471	878	-
	CPU Time	6.50	76.52	13.23
	Optimality	-	0.75	0.78

* The Percentage of LP Solutions

Table 3. Five End Item Problem with Cost Structure 2

		LP Solution	IP Solution	Heuristic Solution
50% and 25% util.	Current Value	107807.6	131497.9	115402.5
	Iterations	698	989	-
	CPU Time	14.62	116.56	20.32
	Optimality*	-	0.81	0.93
75% and 50%	Current Value	107889.5	134606.0	111821.5
	Iterations	600	1250	-
	CPU Time	14.26	125.88	20.32
	Optimality	-	0.80	0.93
95% and 75% util.	Current Value	109095.4	134942.0	114784.3
	Iterations	749	1601	-
	CPU Time	16.18	133.92	20.32
	Optimality	-	0.81	0.93

* The Percentage of LP Solution

REFERENCES

Bahl, H. C. and Ritzman, L. P., 1984, A cyclical scheduling for lot sizing with capacity constraints. International Journal of Production Research,22,791-800.

Billington, P. J., 1983, Multi-Level lot-sizing with a bottleneck work center. Ph.d Dissertation, Cornell University.

Billington, P. J., Blackburn, J. D., Maes, J., Millen, R. A. and Van Wassenhove, L.W., 1988, Multi-product scheduling in multi-stage serial systems. In Chikan, A. and Lovell, M. C. (eds), The Economics of Inventory Management, Elsevier Science, Amsterdam.

Billington, P. J., McLain, J. O. and Thomas, L. J., 1986, Heuristics for multi-level lot-sizing with a bottleneck. Management Science, 32, 989-1006.

Blackburn, J. D. and Millen, R. A., 1982, Improved heuristics for multi-stage requirements planning ystems. Management Science, 28, 44-56.

McLain, J. O. and Trigiero, W. W., 1985, Cylic assembly schedules, IIE Transactions, 17, 346-353.

MGG User Guide, 1987, SD-Scicon, Milton Keynes, England.

Sciconic User Guide, 1986, SD-Scicon, Milton Keynes, England.

Silver, E. A. and Meal, H. C., 1973, A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenisment, Production and Inventory Management, 14, 64-74.

Toklu, B and Wilson J. M., 1991, A heuristic for multilevel lot-sizing problems with a bottleneck, International Journal of Production Research, forthcoming.

LOUGHBOROUGH UNIVERSITY

MANAGEMENT RESEARCH SERIES

1991

No.	Title	Author(s) & Date
1991/		
1.	Quantifying Message Appeals: A Decision Support System for Advertising Strategy	M.A.P. Davies (Jan)
2.	The Best of British: a Peer Evaluation of Britain's Leading Companies	J. Saunders, M. Brown and S. Laverick (Apr)
3.	Developing an Expert Support System for Tender Enquiry Evaluation: a Case Study.	G. Phythian and M. King (Apr)
4.	Validating an Expert Support System for Tender Enquiry Evaluation: a Case Study	G. Phythian and M. King (Apr)
5.	Measures of Success for Management Support Systems	Paul Finlay, (May)
6.	Human Jobs and Computer Interfaces. Figure-ground Reversal in Systems Development and Implementation from HCI to OSI	D. Buchanan (May)
7.	Implementing Competitive Strategy: a People Centred Approach	P. Ahmed and M. Rafiq (May)
8.	Research in Concert: an Evaluation of Publicity	Len Tiu Wright (May)
9.	Marketing Education or Marketing Research?: Alternative Metaphorical Models of the Doctoral Research Process.	Richard Speed (May)

		Author(s) & Date
10.	Validity of Decision support Systems: Simplifiying and Operationalising the Approach.	Paul Finlay and John Wilson (June)
11.	The UK Consumer Market	John Saunders and Jim Saker (July)
12.	Student Accommodation and Its Influence on Student Choice	Mark Davies, Diane Preston and John Wilson (July)
13.	A Survey of Employee Receptivity to Total Quality and its Implications for Training and Programme Achievement	E. Kowalski and P. Walley (Aug)
14.	A Heuristic for Multi Level Lot-Sizing Problems with Multiple Bottleneck	B. Toklu and J. Wilson (Aug)

LOUGHBOROUGH UNIVERSITY

MANAGEMENT RESEARCH SERIES

1990

No.	Title	Author(s) & Date
1990/		
1	Export Controls and Foreign Availability: The Case of Numerical Control	M. R. Hill (January)
2	Computer Aided Design, Manufacture and Production Management in the Soviet Union	M. R. Hill (January)
3	An Evaluation of the Technique of Data Envelopment Analysis in the Measurement of the Relative Efficiencies of University Libraries	B. V. Johnson A. F. Macdougall J. M. Wilson (May)
4	Marketing, Strategy & Performance in the Retail Financial Services Industry: Some Empirical Findings	Richard Speed & Gareth Smith (May)
5	Entry Deterrence Through Entrant Mutual Awareness	Tim James (May)
6	In Search of a Lost Lag: The Quest for a Nest	P. S. H. Leeflang, G. M. Mijatovich & John Saunders (May)
7	Product Replacement Strategies: Occurrences and Coincidences	John Saunders & David Jobber (May)
8	American and Japanese HQ Views of their UK Marketing Operations	Len Wright, John Saunders & Peter Doyle (May)
9	Competition in Global Markets: A Case Study of American and Japanese Competition for the British Market	Peter Doyle, John Saunders & Veronica Wong (May)

No.	Title	Author(s) & Date
10	Appraising Commercial Development Projects: Normative, Descriptive, Investigative and Prescriptive Views	Mohamed Kameshki & John Saunders (May)
11	The Price, Promotion and Quality Decision within Not-For-Profit Organisations	Tim James & John Saunders (June)
12	Are Asians Taking Over British Retailing?	Mohammed Rafiq (June)
13	Ethnicity and Enterprise: Muslim and Non-Muslim owned Asian Businesses in Bradford	Mohammed Rafiq (June)
14	Experiments with Expert Systems in Management Education	Malcolm King & Lawrence McAulay (July)
15	Analyses of Alternative Models in Simple Bidding Situations	Malcolm King (July)
16	Property Performance Measurement	Paul Finlay & Steven Tyler (July)
17	Retail Financial Services Segmentation: Time for New Tricks from Old Dogs?	Richard Speed & Gareth Smith (July)
18	Recruitment Mode as a Factor Affecting Informant Response in Organizational Research	David Buchanan (July)
19	Vulnerability and Agenda: Context and Process in Project Management	David Buchanan (November)
20	Performance Measurement by Expert Assessment in the UK Retail Financial Services Industry	Richard Speed and Gareth Smith (December)
21	Evaluation Techniques Examined and Assessed: a Review of the Literature of Aids to Education	Alison Smith and John Piper (December)

AN ANALYSIS OF MULTIPLE BOTTLENECK PROBLEMS IN MULTILEVEL LOT-SIZING PROBLEMS

by

Bilal Toklu and J M Wilson

A Number of heuristic models have been proposed to determine the lot-sizing techniques for multilevel lot-sizing problems with a bottleneck in order to meet the forecast requirements so far. One of these models involves the use of integer programming using lagrangian relaxation embedded within the branch and bound procedure with one bottleneck. Our previous research to solve the multilevel lot-sizing problem involved the use of a hueristic model rather than the above complex technique.

In this paper we have extended our previous heuristic model to focus upon more than one bottleneck problem. We also compared the economic order quantity technique (EOQ) with the modified Silver Meal technique for non-bottleneck items in multilevel lot- sizing problems, then these results were compared with the integer programming solution. The results have demonstrated that the extended heuristic model is simple to implement and it requires a small fraction of the computer time normally required by the full integer programming approach; the extended heuristic also produces reasonable quality of solutions. The results have also demonstrated that there is no superiority between EOQ and modified Silver Meal techniques for solving this problem.