A heuristic for multi-level lot-sizing problems with bottlenecks under a rolling schedule environment

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/7429

Publisher: © Bilal Toklu

Please cite the published version.
This item is held in Loughborough University’s Institutional Repository (https://dspace.lboro.ac.uk/) and was harvested from the British Library’s EThOS service (http://www.ethos.bl.uk/). It is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
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

© by Bilal Toklu, 1993
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.
CHAPTER 1. INTRODUCTION

1.1. Introduction. .................................................1

CHAPTER 2. REVIEW OF THE LITERATURE

2.1. Introduction. ..................................................5
2.2. The General Lot-Sizing Problem. ............................5
2.2.1. Serial Lot-Sizing Systems .................................9
2.2.2. Non Serial Lot-Sizing Systems ............................14
2.3. Summary. ......................................................23

CHAPTER 3. BACKGROUND WORK

3.1. Introduction. ..................................................24
3.2. Material Requirements Planning. ............................24
3.3. Integer Programming Formulation of Problem. ............27
3.3.1. Assumptions. ...............................................29
3.3.2. Notations. ..................................................30
3.3.3. Mathematical Model. .....................................31
3.4. Summary. ......................................................34
Appendix. .......................................................35
CHAPTER 4. A HEURISTIC FOR MULTI-LEVEL LOT-SIZING PROBLEMS WITH BOTTLENECK(S)

4.1. Purpose of Thesis..........................38
4.2. Introduction...............................39
4.3. Problem Area..............................39
4.4. Simple Heuristic for Multi-Level Lot-Sizing Problem with a Bottleneck...40
  4.4.1. Non-End-Items..........................42
    4.4.1.1. The Economic Order Quantity Approach (EOQ)..................42
    4.4.1.2. The Silver-Meal Approach (SM) .....................43
  4.4.2. End-Items...............................44
4.5. A Heuristic for a Multi-Level Lot-Sizing Problem with Multiple Bottleneck......48
4.6. Sensitivity Analysis.....................51
4.7. Summary and Conclusion..................53

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

5.1. Introduction................................80
5.2. Literature Survey for Rolling Schedule Environment..........................80
5.3. Problem Structure..........................85
5.4. Assumptions...............................87
5.5 Data Sets..................................87
5.6. Simple Heuristic for the Problem with Rolling Schedule.....................88
  5.6.1. Non-End-Items with Bottleneck.................88
5.7. An Example................................90
  5.7.1. Rolling Schedule on Bottleneck Items........91
  5.7.2. Rolling Schedule on Non-Bottleneck Items........92
  5.7.2.1. The Economic Order Quantity Approach......92
  5.7.2.2. The Silver-Meal Approach....................94
5.8. The Results of Rolling Schedule...........95
CHAPTER 6. RESULTS AND DISCUSSIONS

6.1. Introduction........................................... 103
6.2. The Results and Discussions of Multi-Level Lot-Sizing Problem with Bottleneck(s) ... 104
6.3. The Results and Discussions of Rolling Schedule for Multi-Level Lot-Sizing Problem with Bottleneck(s) .............. 110
6.4. Worst Case Analysis of Heuristics................. 112
6.5. Summary and Conclusion......................... 115

CHAPTER 7. A COMPARISON FOR THE ASSEMBLY SYSTEM

7.1. Introduction........................................... 117
7.2. A Comparison for Assembly Systems......... 117
7.3. Sensitivity Analysis of the Heuristics.. 124
7.4. Conclusion........................................... 126

CHAPTER 8 CONCLUSIONS AND FUTURE RESEARCH

8.1. Introduction........................................... 127
8.2. Conclusions........................................... 127
8.3. Recommendations for the Future Research.129

REFERENCES.................................................. 131

Appendix A. Model in MGG on SCICONIC................. 142
Appendix B. Data.......................................... 146
Appendix C. Heuristic Program on Fortran 77 ........ 153
Appendix D. NAG Subroutine Program..................... 186
Appendix E. Reprint of Published Papers............. 188
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6.</td>
<td>Sensitivity Analysis for One-End-Item Problem</td>
<td>53</td>
</tr>
<tr>
<td>4.6.1.</td>
<td>One End Item Problem with Structure One</td>
<td>56</td>
</tr>
<tr>
<td>4.6.2.</td>
<td>One End Item Problem with Structure Two</td>
<td>57</td>
</tr>
<tr>
<td>4.6.3.</td>
<td>One End Item Problem with Structure Three</td>
<td>59</td>
</tr>
<tr>
<td>4.6.4.</td>
<td>Three End Item Problem with Structure One</td>
<td>60</td>
</tr>
<tr>
<td>4.6.5.</td>
<td>Three End Item Problem with Structure Two</td>
<td>62</td>
</tr>
<tr>
<td>4.6.6.</td>
<td>Three End Item Problem with Structure Three</td>
<td>63</td>
</tr>
<tr>
<td>4.6.7.</td>
<td>Five End Item Problem with Structure One</td>
<td>65</td>
</tr>
<tr>
<td>4.6.8.</td>
<td>Five End Item Problem with Structure Two</td>
<td>66</td>
</tr>
<tr>
<td>4.6.9.</td>
<td>Five End Item Problem with Structure Three</td>
<td>68</td>
</tr>
<tr>
<td>4.6.10.</td>
<td>One End Item Problem with Structure One</td>
<td>70</td>
</tr>
<tr>
<td>4.6.11.</td>
<td>Three End Item Problem with Structure Two</td>
<td>71</td>
</tr>
<tr>
<td>4.6.12.</td>
<td>Five End Item Problem with Structure Three</td>
<td>73</td>
</tr>
<tr>
<td>4.6.13.</td>
<td>One End Item Problem with Structure One</td>
<td>75</td>
</tr>
<tr>
<td>4.6.14.</td>
<td>One End Item Problem with Structure Three</td>
<td>75</td>
</tr>
<tr>
<td>4.6.15.</td>
<td>Three End Item Problem with Structure One</td>
<td>76</td>
</tr>
<tr>
<td>4.6.16.</td>
<td>Three End Item Problem with Structure Three</td>
<td>76</td>
</tr>
<tr>
<td>4.6.17.</td>
<td>Five End Item Problem with Structure One</td>
<td>77</td>
</tr>
<tr>
<td>4.6.18.</td>
<td>Five End Item Problem with Structure Three</td>
<td>77</td>
</tr>
<tr>
<td>4.6.19.</td>
<td>One End Item Problem with Structure One</td>
<td>78</td>
</tr>
<tr>
<td>4.6.20.</td>
<td>Three End Item Problem with Structure Two</td>
<td>78</td>
</tr>
<tr>
<td>4.6.21.</td>
<td>Five End Item Problem with Structure Three</td>
<td>79</td>
</tr>
<tr>
<td>5.8.1.</td>
<td>One End Item Problem with Structure One</td>
<td>98</td>
</tr>
<tr>
<td>5.8.2.</td>
<td>One End Item Problem with Structure Two</td>
<td>98</td>
</tr>
<tr>
<td>5.8.3.</td>
<td>One End Item Problem with Structure Three</td>
<td>99</td>
</tr>
<tr>
<td>5.8.4.</td>
<td>Three End Item Problem with Structure One</td>
<td>99</td>
</tr>
<tr>
<td>5.8.5.</td>
<td>Three End Item Problem with Structure Two</td>
<td>100</td>
</tr>
<tr>
<td>5.8.6.</td>
<td>Three End Item Problem with Structure Three</td>
<td>100</td>
</tr>
<tr>
<td>5.8.7.</td>
<td>Five End Item Problem with Structure One</td>
<td>101</td>
</tr>
<tr>
<td>5.8.8.</td>
<td>Five End Item Problem with Structure Two</td>
<td>101</td>
</tr>
<tr>
<td>5.8.9.</td>
<td>Five End Item Problem with Structure Three</td>
<td>102</td>
</tr>
<tr>
<td>6.4.</td>
<td>A Cost Analysis</td>
<td>114</td>
</tr>
<tr>
<td>7.1.</td>
<td>Data for a 5-period problem</td>
<td>120</td>
</tr>
</tbody>
</table>
7.2. Demand and capacity for a 5-period problem...121
7.3. Schedules and costs for the 4-period problem.122
7.4. Schedules and costs for the 4-period problem.122
7.5. Final solution for the 5-period problem......123
7.6. Solution using proposed heuristic.............124
7.7. Total Cost under Different Cost Assumptions..125
LIST OF FIGURES

Figure
2.2. Different Product Types ......................... 15
3.1. A General Product Product Structure with a
    Bottleneck Facility ............................ 28
4.4. A Parallel Product Structure with Multiple
    Bottleneck Facilities .......................... 49
5.3. Five End Item structure ........................ 86
7.1. The three stage assembly systems ............... 121
LIST OF GRAPHS

Graph

2.2. Total Cost versus Lot-Size ..................... 7

The following graphs are on pages .................. 79-80

4.6.1. One End Item Problem with Structure 1
4.6.2. One End Item Problem with Structure 1
4.6.3. One End Item Problem with Structure 1
4.6.4. Three End Item Problem with Structure 1
4.6.5. Three End Item Problem with Structure 1
4.6.6. Three End Item Problem with Structure 1
4.6.7. Five End Item Problem with Structure 1
4.6.8. Five End Item Problem with Structure 1
4.6.9. Five End Item Problem with Structure 1
4.6.10. One End Item Problem with Structure 1
4.6.11. Three End Item Problem with Structure 2
4.6.12. Five End Item Problem with Structure 3

The following graphs are on pages .................. 102-103

5.8.1. One End Item Problem with Structure 1
5.8.2. One End Item Problem with Structure 2
5.8.3. One End Item Problem with Structure 3
5.8.4. Three End Item Problem with Structure 1
5.8.5. Three End Item Problem with Structure 2
5.8.6. Three End Item Problem with Structure 3
5.8.7. Five End Item Problem with Structure 1
5.8.8. Five End Item Problem with Structure 2
5.8.9. Five End Item Problem with Structure 3

6.4. The Relation Between Setup and Holding Costs 115
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 addresses 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 points. 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


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


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])

\[
\text{(P)} \quad \text{Minimise} \quad \sum_{j=1}^{N} \left[ \frac{s_j}{n_j} + \frac{h_jD_jP(j)(k_j^{-1})}{2} \right]
\]
Subject to

\[ n_j = k_j n_{p(j)} \quad j = 2, \ldots, N \]

\[ n_j \leq \frac{V_j}{D_j} \quad j = 1, \ldots, M \]

\[ n_j, k_j \geq 1 \text{ and integer} \]

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

\[ a_{ij} = \text{The quantity of product } i \text{ needed per unit of production of product } j; \quad a_{ij} = 0 \text{ for all } i < j, \]
\[ b_i = \text{Time needed on the bottleneck facility for the production of product } i, \text{ and } b_i = 0 \text{ if there is no production on the bottleneck facility}, \]
\[ \text{cap}_t = \text{Units of time available at the work centre}, \]
\[ c_{si} = \text{Setup cost for product } i \text{ (following the assumption made by Billington et al. [1986], for consistency, the possibility of "carrying over" a setup from one period to another is not allowed)}, \]
\[ d_{it} = \text{External (independent) demand for product } i \text{ during time } t, \text{ and there is no demand for the intermediate items,} \]

\[ h_i = \text{Holding cost for product } i, \]

\[ I_{it} = \text{Final inventory level of product } i \text{ in period } t, \]

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

\[ N = \text{Number of products,} \]

\[ P_{it} = \text{Units of } i \text{ produced in period } t, \]

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

\[ T = \text{Total number of periods,} \]

\[ X_{it} = \text{Production indicator; equal to 1 if } P_{it} > 0 \text{ and zero otherwise.} \]

### 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:

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

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:

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

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).

\[
\sum_{i=1}^{N} \left[ b_i P_{it} + s_i X_{it} \right] \leq \text{cap}_t \quad t=1, \ldots, T
\]

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.

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

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

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

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 closely 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)

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

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

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

therefore holding cost
\[= QTh_i + Q \left( T - \frac{QT}{D} \right) h_i + Q \left( T - \frac{2QT}{D} \right) h_i + \]

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

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

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

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

\[S = \frac{n}{2} \left( 2a + (n - 1) d \right) \]

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

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

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

therefore holding cost
Total cost of setup and holding = $TC$

\[
= \frac{D_{csi}}{Q} + QThi \left[ \frac{D}{Q} - \frac{1}{2} \left( \frac{D}{Q} - 1 \right) \right]
\]

Differentiating with respect to $Q$:

\[
\frac{dTC}{dQ} = -\frac{D_{csi}}{Q^2} + \frac{Thi}{2}
\]

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

Minimum cost batch size $= Q^* = \sqrt{\frac{2D_{csi}}{Thi}}$

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

therefore $Q^* = \sqrt{\frac{2d_{csi}}{h_i}}$

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 successful 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 $t$

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

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 $c_{si}$ 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

$$t_i = \left\lfloor \sum_{n=1}^{T} d_{it} / Q_i + 0.5 \right\rfloor$$

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

Note that $\lfloor \ \rfloor$ is the integer part function.
If in any period \( n \)

\[ Q_i \geq \sum_{t=n}^{T} d_i t - S_{in} \]

then set \( Q_i = \sum_{t=n}^{T} d_i t - S_{in} \)

(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 \( S_{in} < d_{in} \)).

\[ Z_i = \left( \sum_{t=1}^{T} d_i t \right) / t_i \]

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:

\[ \text{Average total Cost} = \frac{\sum_{t=1}^{T} (c_{si} + h_i \sum_{t=1}^{t-1} (t-1) d_{it})}{t} \]

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 \( \text{cap}_t \) will allow in period 1, i.e. set \( P_{i1} = \text{cap}_1 \), 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 $P_{11} = \text{cap}_1 - d_{21} - d_{31}$.

**Period 2** Set $P_{32} = d_{32}$, $P_{12} = 0$ and $P_{22} = \text{cap}_2 - d_{32}$ provided $S_{12} > d_{12}$.

Otherwise set $P_{12} = d_{12} + d_{13} - S_{12}$, $P_{32} = d_{32}$ and $P_{22} = \text{cap}_2 - P_{12} - P_{32}$.

**Period 3** Set $P_{33} = \text{cap}_3$ 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 $P_{33} = \text{cap}_3 - P_{13} - P_{23}$.

**Period 4** Set $P_{14} = \text{cap}_4$ provided $S_{24} > d_{24}$ and $S_{34} > d_{34}$.

**Period 5** Set $P_{25} = \text{cap}_5$ provided $S_{15} > d_{15}$ and $S_{35} > d_{35}$.

**Period 6** Set $P_{36} = \text{cap}_6$ 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 \( P_{11} = \text{cap}_1 - P_{41} \).

Period 2 Set \( P_{12} = 0 \) and \( P_{42} = \text{cap}_2 \) provided \( S_{12} > d_{12} \) Otherwise set \( P_{12} = d_{12} + d_{13} - S_{42}, P_{42} = \text{cap}_2 - P_{12} \)

Period 3 Set \( P_{13} = \text{cap}_3 \) provided \( S_{43} > d_{43} \)

Period 4 Set \( P_{44} = \text{cap}_4 \) 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:

\[ Z = \sum_{i=1}^{N} \sum_{t=1}^{T} \left( h_i (T - t + 1) P_{it} + c_{si} X_{it} \right) \]

Subject to:

\[ \sum_{n=1}^{t} \left[ P_{i,n-1} - \sum_{j=1}^{t} a_{ij} P_{jn} \right] \geq \sum_{n=1}^{t} d_{in} - I_{io} \quad i=1, \ldots, N \quad t=1, \ldots, T \]
\[ \sum_{i=1}^{N} \left( b_{il} P_{it} + s_{il} X_{it} \right) \leq c_{at} \quad t=1, \ldots, T \]

\[ i=1, \ldots, M \]

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

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

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 $P_{71} = c_{ap1} - d_{81} - d_{91}$.

Period 2  Set $P_{92} = d_{92}$, $P_{72} = 0$ and $P_{82} = c_{ap2} - d_{92}$ provided $S_{72} > d_{72}$.
Otherwise set $P_{72} = d_{72} + d_{73} - S_{72}$, $P_{92} = d_{92}$ and $P_{82} = c_{ap2} - P_{72} - P_{92}$.

Period 3  Set $P_{93} = c_{ap3}$ 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 $P_{93} = c_{ap3} - P_{73} - P_{83}$.

Period 4  Set $P_{74} = c_{ap4}$ provided $S_{84} > d_{84}$ and $S_{94} > d_{94}$.

Period 5  Set $P_{85} = c_{ap5}$ provided $S_{75} > d_{75}$ and $S_{95} > d_{95}$.

Period 6  Set $P_{96} = c_{ap6}$ 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

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

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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EOQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>Current Value</td>
<td>10869</td>
<td>15441</td>
<td>13070</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>144</td>
<td>3332</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.80</td>
<td>127.98</td>
<td>1.20</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.420</td>
<td>1.202</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.212</td>
</tr>
</tbody>
</table>

Optimality* = Current Value/LP Current Value

Table 4.6.1. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EOQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>Current Value</td>
<td>8695</td>
<td>12352</td>
<td>10456</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>144</td>
<td>3332</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.78</td>
<td>129.36</td>
<td>1.20</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.420</td>
<td>1.202</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.212</td>
</tr>
</tbody>
</table>

Optimality* = Current Value/LP Current Value

** Calculated for a 20% reduction in the Original Costs
Table 4.6.1. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50%</td>
<td>75%</td>
<td>95%</td>
</tr>
<tr>
<td>Current Value</td>
<td>6521</td>
<td>6538</td>
<td>6521</td>
</tr>
<tr>
<td>Iterations</td>
<td>144</td>
<td>151</td>
<td>148</td>
</tr>
<tr>
<td>Util.</td>
<td>0.76</td>
<td>0.76</td>
<td>0.90</td>
</tr>
<tr>
<td>CPU Time</td>
<td>0.76</td>
<td>1.01</td>
<td>0.80</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Current Value</td>
<td>9264</td>
<td>9606</td>
<td>9626</td>
</tr>
<tr>
<td>Iterations</td>
<td>3332</td>
<td>1684</td>
<td>2393</td>
</tr>
<tr>
<td>Util.</td>
<td>127.18</td>
<td>130.68</td>
<td>101.02</td>
</tr>
<tr>
<td>CPU Time</td>
<td>1.21</td>
<td>1.21</td>
<td>1.22</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.20</td>
<td>1.28</td>
<td>1.22</td>
</tr>
<tr>
<td>Current Value</td>
<td>7842</td>
<td>8386</td>
<td>8888</td>
</tr>
<tr>
<td>Iterations</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util.</td>
<td>1.21</td>
<td>1.21</td>
<td>1.22</td>
</tr>
<tr>
<td>CPU Time</td>
<td>1.20</td>
<td>1.28</td>
<td>1.22</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.20</td>
<td>1.28</td>
<td>1.22</td>
</tr>
<tr>
<td>Current Value</td>
<td>7910</td>
<td>8454</td>
<td>8956</td>
</tr>
<tr>
<td>Iterations</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util.</td>
<td>1.21</td>
<td>1.21</td>
<td>1.22</td>
</tr>
<tr>
<td>CPU Time</td>
<td>1.21</td>
<td>1.28</td>
<td>1.22</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.20</td>
<td>1.28</td>
<td>1.22</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>30036</td>
<td>36235</td>
<td>35785</td>
</tr>
<tr>
<td>Iteration</td>
<td>144</td>
<td>1713</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.80</td>
<td>60.36</td>
<td>3.10</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.206</td>
<td>1.191</td>
</tr>
<tr>
<td>Current Value</td>
<td>30036</td>
<td>38110</td>
<td>37293</td>
</tr>
<tr>
<td>Iteration</td>
<td>154</td>
<td>2420</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.88</td>
<td>78.92</td>
<td>3.11</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.268</td>
<td>1.241</td>
</tr>
<tr>
<td>Current Value</td>
<td>30036</td>
<td>38498</td>
<td>38700</td>
</tr>
<tr>
<td>Iteration</td>
<td>150</td>
<td>1828</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.84</td>
<td>78.92</td>
<td>3.11</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.269</td>
<td>1.275</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
Table 4.6.2. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution **</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current Value</td>
<td>Iterations</td>
<td>CPU Time</td>
</tr>
<tr>
<td>50%</td>
<td>24029</td>
<td>144</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>28988</td>
<td>1713</td>
<td>60.45</td>
</tr>
<tr>
<td></td>
<td>28628</td>
<td>-</td>
<td>3.10</td>
</tr>
<tr>
<td></td>
<td>27799</td>
<td>-</td>
<td>3.10</td>
</tr>
<tr>
<td>75%</td>
<td>24029</td>
<td>154</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>30488</td>
<td>2420</td>
<td>79.66</td>
</tr>
<tr>
<td></td>
<td>29834</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td></td>
<td>29005</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td>95%</td>
<td>24265</td>
<td>150</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>30798</td>
<td>1828</td>
<td>60.74</td>
</tr>
<tr>
<td></td>
<td>30960</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td></td>
<td>30131</td>
<td>-</td>
<td>3.11</td>
</tr>
</tbody>
</table>

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

Table 4.6.2. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current Value</td>
<td>Iterations</td>
<td>CPU Time</td>
</tr>
<tr>
<td>50%</td>
<td>18021</td>
<td>144</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>21741</td>
<td>1713</td>
<td>60.36</td>
</tr>
<tr>
<td></td>
<td>21471</td>
<td>-</td>
<td>3.10</td>
</tr>
<tr>
<td></td>
<td>20849</td>
<td>-</td>
<td>3.10</td>
</tr>
<tr>
<td>75%</td>
<td>18021</td>
<td>154</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>22866</td>
<td>1796</td>
<td>79.41</td>
</tr>
<tr>
<td></td>
<td>22375</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td></td>
<td>21754</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td>95%</td>
<td>18198</td>
<td>150</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>23099</td>
<td>1828</td>
<td>60.20</td>
</tr>
<tr>
<td></td>
<td>23220</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td></td>
<td>22598</td>
<td>-</td>
<td>3.11</td>
</tr>
</tbody>
</table>

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

58
Table 4.6.3. One End Item Problem with Structure Three

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EOQ</td>
</tr>
<tr>
<td><strong>Sol.</strong></td>
<td>18787</td>
<td>26102</td>
<td>23716</td>
</tr>
<tr>
<td><strong>50%</strong></td>
<td>150</td>
<td>2338</td>
<td>-</td>
</tr>
<tr>
<td><strong>Iter.</strong></td>
<td>0.94</td>
<td>77.23</td>
<td>3.12</td>
</tr>
<tr>
<td><strong>Util.</strong></td>
<td>1.000</td>
<td>1.389</td>
<td>1.262</td>
</tr>
<tr>
<td><strong>Optimal.</strong></td>
<td>1.000</td>
<td>1.389</td>
<td>1.262</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current Value</th>
<th>18863</th>
<th>26434</th>
<th>24596</th>
<th>23867</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>50%</strong></td>
<td>147</td>
<td>1832</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Iter.</strong></td>
<td>0.76</td>
<td>50.52</td>
<td>3.12</td>
<td>3.12</td>
</tr>
<tr>
<td><strong>Util.</strong></td>
<td>1.000</td>
<td>1.401</td>
<td>1.303</td>
<td>1.265</td>
</tr>
<tr>
<td><strong>Optimal.</strong></td>
<td>1.000</td>
<td>1.401</td>
<td>1.303</td>
<td>1.265</td>
</tr>
</tbody>
</table>

| **Current Value** | 19375 | 27378 | 25502 | 24773 |
| **75%**           | 141   | 2258  | -     | -     |
| **Iter.**         | 0.84  | 65.65 | 3.13  | 3.13  |
| **Util.**         | 1.000 | 1.413 | 1.316 | 1.278 |
| **Optimal.**      | 1.000 | 1.413 | 1.316 | 1.278 |

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

Table 4.6.3. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EOQ</td>
</tr>
<tr>
<td><strong>Sol.</strong></td>
<td>15030</td>
<td>20882</td>
<td>18972</td>
</tr>
<tr>
<td><strong>50%</strong></td>
<td>150</td>
<td>2338</td>
<td>-</td>
</tr>
<tr>
<td><strong>Iter.</strong></td>
<td>0.86</td>
<td>77.92</td>
<td>3.12</td>
</tr>
<tr>
<td><strong>Util.</strong></td>
<td>1.000</td>
<td>1.389</td>
<td>1.262</td>
</tr>
<tr>
<td><strong>Optimal.</strong></td>
<td>1.000</td>
<td>1.389</td>
<td>1.262</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current Value</th>
<th>15090</th>
<th>21147</th>
<th>19677</th>
<th>19093</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>75%</strong></td>
<td>147</td>
<td>1832</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Iter.</strong></td>
<td>0.78</td>
<td>50.59</td>
<td>3.13</td>
<td>3.13</td>
</tr>
<tr>
<td><strong>Util.</strong></td>
<td>1.000</td>
<td>1.401</td>
<td>1.303</td>
<td>1.265</td>
</tr>
<tr>
<td><strong>Optimal.</strong></td>
<td>1.000</td>
<td>1.401</td>
<td>1.303</td>
<td>1.265</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current Value</th>
<th>15500</th>
<th>21902</th>
<th>20401</th>
<th>19818</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>95%</strong></td>
<td>141</td>
<td>2558</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Iter.</strong></td>
<td>0.78</td>
<td>66.98</td>
<td>3.13</td>
<td>3.13</td>
</tr>
<tr>
<td><strong>Util.</strong></td>
<td>1.000</td>
<td>1.413</td>
<td>1.316</td>
<td>1.278</td>
</tr>
<tr>
<td><strong>Optimal.</strong></td>
<td>1.000</td>
<td>1.413</td>
<td>1.316</td>
<td>1.278</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% reduction in the Original Costs
### Table 4.6.3. Continued

|           | 50% Iterations | 75% Iterations | 95% Iterations | Util. CPU Time | Optimality* | 50% Iterations | 75% Iterations | 95% Iterations | Util. CPU Time | Optimality* | 50% Iterations | 75% Iterations | 95% Iterations | Util. CPU Time | Optimality* |
|-----------|----------------|----------------|----------------|----------------|-------------|----------------|----------------|----------------|----------------|-------------|----------------|----------------|----------------|----------------|-------------|-------------|
| Current Value | 11272          | 15661          | 14229          | 13792          |             | 150            | 2338           | --             | --             |             | 0.88           | 77.45          | 3.12           | 3.12          |             |
|              | 150            | 2338           | --             | --             |             |                |                |                |                |             |                |                |                |                |             |
| Util. CPU Time | 0.88           | 77.45          | 3.12           | 3.12           |             |                |                |                |                |             |                |                |                |                |             |
| Optimality* | 1.000          | 1.389          | 1.262          | 1.223          |             |                |                |                |                |             |                |                |                |                |             |
| Current Value | 11318          | 15860          | 14757          | 14320          |             | 147            | 1832           | --             | --             |             | 0.86           | 50.85          | 3.13           | 3.13          |             |
|              | 147            | 1832           | --             | --             |             |                |                |                |                |             |                |                |                |                |             |
| Util. CPU Time | 0.86           | 50.85          | 3.13           | 3.13           |             |                |                |                |                |             |                |                |                |                |             |
| Optimality* | 1.000          | 1.401          | 1.303          | 1.265          |             |                |                |                |                |             |                |                |                |                |             |

* 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

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

* The proportion is calculated as Current Value/LP Current Value
Table 4.6.4. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>**</td>
<td>EOQ</td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Current Value</td>
<td>58898</td>
<td>73666</td>
<td>69725</td>
</tr>
<tr>
<td>50%</td>
<td>Iterations</td>
<td>430</td>
<td>1503</td>
</tr>
<tr>
<td>Util.</td>
<td>CPU Time</td>
<td>5.69</td>
<td>368.22</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.250</td>
<td>1.183</td>
</tr>
<tr>
<td>Current Value</td>
<td>58898</td>
<td>74458</td>
<td>70119</td>
</tr>
<tr>
<td>75%</td>
<td>Iterations</td>
<td>445</td>
<td>1807</td>
</tr>
<tr>
<td>Util.</td>
<td>CPU Time</td>
<td>5.56</td>
<td>341.39</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.264</td>
<td>1.190</td>
</tr>
<tr>
<td>Current Value</td>
<td>59121</td>
<td>75234</td>
<td>73487</td>
</tr>
<tr>
<td>95%</td>
<td>Iterations</td>
<td>460</td>
<td>769</td>
</tr>
<tr>
<td>Util.</td>
<td>CPU Time</td>
<td>6.06</td>
<td>351.54</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.272</td>
<td>1.242</td>
</tr>
</tbody>
</table>

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

Table 4.6.4. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>**</td>
<td>EOQ</td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Current Value</td>
<td>44174</td>
<td>55249</td>
<td>52294</td>
</tr>
<tr>
<td>50%</td>
<td>Iterations</td>
<td>430</td>
<td>1503</td>
</tr>
<tr>
<td>Util.</td>
<td>CPU Time</td>
<td>5.28</td>
<td>319.18</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.250</td>
<td>1.183</td>
</tr>
<tr>
<td>Current Value</td>
<td>44174</td>
<td>55843</td>
<td>52589</td>
</tr>
<tr>
<td>75%</td>
<td>Iterations</td>
<td>445</td>
<td>1807</td>
</tr>
<tr>
<td>Util.</td>
<td>CPU Time</td>
<td>5.67</td>
<td>353.58</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.264</td>
<td>1.190</td>
</tr>
<tr>
<td>Current Value</td>
<td>44341</td>
<td>56426</td>
<td>55115</td>
</tr>
<tr>
<td>95%</td>
<td>Iterations</td>
<td>460</td>
<td>769</td>
</tr>
<tr>
<td>Util.</td>
<td>CPU Time</td>
<td>6.20</td>
<td>332.94</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.272</td>
<td>1.242</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
*** Calculated for a 40% reduction in the Original Costs
<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EOQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Silver-Meal</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>78353</td>
<td>97496</td>
<td>99336</td>
</tr>
<tr>
<td><strong>50% Iterations</strong></td>
<td>427</td>
<td>1039</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.22</td>
<td>312.01</td>
<td>3.14</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.244</td>
<td>1.226</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>78367</td>
<td>98608</td>
<td>100086</td>
</tr>
<tr>
<td><strong>75% Iterations</strong></td>
<td>435</td>
<td>663</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.78</td>
<td>325.9</td>
<td>3.15</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.258</td>
<td>1.277</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>79163</td>
<td>102507</td>
<td>102629</td>
</tr>
<tr>
<td><strong>95% Iterations</strong></td>
<td>438</td>
<td>3625</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.96</td>
<td>423.32</td>
<td>3.16</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.294</td>
<td>1.296</td>
</tr>
</tbody>
</table>

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

Table 4.6.5. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EOQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Silver-Meal</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>62683</td>
<td>77997</td>
<td>79469</td>
</tr>
<tr>
<td><strong>50% Iterations</strong></td>
<td>427</td>
<td>1037</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.34</td>
<td>342.8</td>
<td>3.15</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.244</td>
<td>1.267</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>62694</td>
<td>78886</td>
<td>80069</td>
</tr>
<tr>
<td><strong>75% Iterations</strong></td>
<td>435</td>
<td>663</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.84</td>
<td>338.8</td>
<td>3.15</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.258</td>
<td>1.277</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>63330</td>
<td>82006</td>
<td>82103</td>
</tr>
<tr>
<td><strong>95% Iterations</strong></td>
<td>435</td>
<td>3610</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.80</td>
<td>412.42</td>
<td>3.16</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.294</td>
<td>1.296</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
** Calculated for a 20% reduction in the Original Costs
Table 4.6.5. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current Value</td>
<td>47012</td>
<td>58498</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>427</td>
<td>1037</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.70</td>
<td>374.2</td>
<td>3.15</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.244</td>
<td>1.267</td>
</tr>
<tr>
<td></td>
<td>Current Value</td>
<td>47020</td>
<td>59165</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>435</td>
<td>663</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.86</td>
<td>377.98</td>
<td>3.16</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.258</td>
<td>1.277</td>
</tr>
<tr>
<td></td>
<td>Current Value</td>
<td>47498</td>
<td>61462</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>433</td>
<td>4188</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>6.02</td>
<td>443.72</td>
<td>3.16</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.294</td>
<td>1.296</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current Value</td>
<td>58424</td>
<td>75478</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>393</td>
<td>835</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.14</td>
<td>359.00</td>
<td>3.16</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.291</td>
<td>1.197</td>
</tr>
<tr>
<td></td>
<td>Current Value</td>
<td>59061</td>
<td>74549</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>431</td>
<td>2238</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.74</td>
<td>375.3</td>
<td>3.18</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.262</td>
<td>1.176</td>
</tr>
<tr>
<td></td>
<td>Current Value</td>
<td>61973</td>
<td>75723</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>433</td>
<td>1713</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.78</td>
<td>458.51</td>
<td>3.18</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.221</td>
<td>1.128</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
### Table 4.6.6. Continued

<table>
<thead>
<tr>
<th></th>
<th>Current Value</th>
<th>50% Iterations</th>
<th>Util. CPU Time</th>
<th>Optimality*</th>
<th>Current Value</th>
<th>75% Iterations</th>
<th>Util. CPU Time</th>
<th>Optimality*</th>
<th>Current Value</th>
<th>95% Iterations</th>
<th>Util. CPU Time</th>
<th>Optimality*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LP Sol.</strong></td>
<td>46739</td>
<td>393</td>
<td>4.98</td>
<td>1.000</td>
<td>47249</td>
<td>431</td>
<td>5.76</td>
<td>1.000</td>
<td>49579</td>
<td>433</td>
<td>5.64</td>
<td>1.000</td>
</tr>
<tr>
<td><strong>IP Sol.</strong></td>
<td>60382</td>
<td>835</td>
<td>279.5</td>
<td>1.291</td>
<td>59639</td>
<td>2238</td>
<td>370.48</td>
<td>1.262</td>
<td>55572</td>
<td>1705</td>
<td>454.35</td>
<td>1.221</td>
</tr>
<tr>
<td><strong>Heuristic Solution</strong></td>
<td>55991</td>
<td>55020</td>
<td>3.17</td>
<td>1.197</td>
<td>55591</td>
<td>2238</td>
<td>3.18</td>
<td>1.176</td>
<td>54601</td>
<td>1705</td>
<td>3.18</td>
<td>1.155</td>
</tr>
<tr>
<td><strong>EOQ Silver-Meal</strong></td>
<td>55020</td>
<td>55020</td>
<td>3.17</td>
<td>1.177</td>
<td>54596</td>
<td>2238</td>
<td>3.18</td>
<td>1.155</td>
<td>54996</td>
<td>1705</td>
<td>3.18</td>
<td>1.109</td>
</tr>
</tbody>
</table>

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

### Table 4.6.6. Continued

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

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EOQ</strong></td>
<td><strong>Silver-Meal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>128948</td>
<td>158390</td>
<td>136935</td>
</tr>
<tr>
<td>50% <strong>Iterations</strong></td>
<td>699</td>
<td>2797</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>14.50</td>
<td>675.85</td>
<td>1.07</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.228</td>
<td>1.061</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>128948</td>
<td>159198</td>
<td>136838</td>
</tr>
<tr>
<td>75% <strong>Iterations</strong></td>
<td>721</td>
<td>2864</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>14.19</td>
<td>592.60</td>
<td>1.08</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.234</td>
<td>1.061</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>129108</td>
<td>162936</td>
<td>137141</td>
</tr>
<tr>
<td>95% <strong>Iterations</strong></td>
<td>774</td>
<td>3825</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>16.28</td>
<td>684.9</td>
<td>1.09</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.262</td>
<td>1.062</td>
</tr>
</tbody>
</table>

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

### Table 4.6.7. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EOQ</strong></td>
<td><strong>Silver-Meal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>103158</td>
<td>126712</td>
<td>109548</td>
</tr>
<tr>
<td>50% <strong>Iterations</strong></td>
<td>699</td>
<td>3156</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>14.02</td>
<td>665.00</td>
<td>1.08</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.228</td>
<td>1.061</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>103158</td>
<td>127358</td>
<td>109470</td>
</tr>
<tr>
<td>75% <strong>Iterations</strong></td>
<td>721</td>
<td>3486</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>14.16</td>
<td>573.24</td>
<td>1.08</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.234</td>
<td>1.061</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>103286</td>
<td>130348</td>
<td>109713</td>
</tr>
<tr>
<td>95% <strong>Iterations</strong></td>
<td>774</td>
<td>3825</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>15.92</td>
<td>681.7</td>
<td>1.09</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.262</td>
<td>1.062</td>
</tr>
</tbody>
</table>

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

** Calculated for a 20% reduction in the Original Costs
Table 4.6.7. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>77369</td>
<td>95034</td>
<td>82161</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>699</td>
<td>3156</td>
<td>79904</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>14.18</td>
<td>612.14</td>
<td>1.08</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.228</td>
<td>1.08</td>
</tr>
<tr>
<td>Current Value</td>
<td>95518</td>
<td>82102</td>
<td>79845</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>721</td>
<td>3486</td>
<td></td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>14.10</td>
<td>566.78</td>
<td>1.09</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.234</td>
<td>1.09</td>
</tr>
<tr>
<td>Current Value</td>
<td>97761</td>
<td>82284</td>
<td>80027</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>774</td>
<td>3825</td>
<td></td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>15.72</td>
<td>680.54</td>
<td>1.09</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.262</td>
<td>1.09</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>107807</td>
<td>131277</td>
<td>108211</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>697</td>
<td>2610</td>
<td></td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>13.69</td>
<td>596.20</td>
<td>2.35</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.217</td>
<td>1.003</td>
</tr>
<tr>
<td>Current Value</td>
<td>107899</td>
<td>134606</td>
<td>108853</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>685</td>
<td>1216</td>
<td></td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>13.97</td>
<td>751.6</td>
<td>2.36</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.247</td>
<td>1.008</td>
</tr>
<tr>
<td>Current Value</td>
<td>109141</td>
<td>134673</td>
<td>111845</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>694</td>
<td>3526</td>
<td></td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>15.12</td>
<td>665.65</td>
<td>2.36</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.233</td>
<td>1.024</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
~ Taking into account the rounding error
<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td>Current Value</td>
<td>86246</td>
<td>105022</td>
<td>86569</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>697</td>
<td>2610</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>14.06</td>
<td>612.9</td>
<td>2.35</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.217</td>
<td>1.003</td>
</tr>
<tr>
<td></td>
<td>86311</td>
<td>107684</td>
<td>87082</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>685</td>
<td>1216</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>14.30</td>
<td>732.5</td>
<td>2.36</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.247</td>
<td>1.008</td>
</tr>
<tr>
<td></td>
<td>87313</td>
<td>107657</td>
<td>89476</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>694</td>
<td>4445</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>15.33</td>
<td>639.77</td>
<td>2.37</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.233</td>
<td>1.024</td>
</tr>
</tbody>
</table>

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

* 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

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

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

~ Taking into account the rounding error

### Table 4.6.9. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>76481</td>
<td>96025</td>
<td>80306</td>
</tr>
<tr>
<td><strong>50% Iterations</strong></td>
<td>736</td>
<td>3295</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>14.04</td>
<td>523.52</td>
<td>1.24</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.255</td>
<td>1.050</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>77377</td>
<td>97660</td>
<td>80322</td>
</tr>
<tr>
<td><strong>75% Iterations</strong></td>
<td>713</td>
<td>1298</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>14.02</td>
<td>620.02</td>
<td>1.24</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.262</td>
<td>1.038</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>79673</td>
<td>100037</td>
<td>81140</td>
</tr>
<tr>
<td><strong>95% Iterations</strong></td>
<td>751</td>
<td>3223</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>16.34</td>
<td>648.19</td>
<td>1.25</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.255</td>
<td>1.018</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP</th>
<th>IP</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sol.***</td>
<td>Sol.***</td>
<td>EOQ</td>
</tr>
<tr>
<td>Current Value</td>
<td>57361</td>
<td>72019</td>
<td>60229</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>713</td>
<td>3230</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>13.94</td>
<td>508.58</td>
<td>1.24</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.255</td>
<td>1.050</td>
</tr>
</tbody>
</table>

|                | Current Value | 58033 | 73245 | 60242 | 57476 |
| 75% Iterations | 702           | 1292  | -    | -    |
| Util. CPU Time | 13.52         | 615.05| 1.24 | 1.24 |
| Optimality*    | 1.000         | 1.262 | 1.038| 1.000~|

|                | Current Value | 59754 | 75028 | 60855 | 58090 |
| 95% Iterations | 750           | 3270  | -    | -    |
| Util. 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td>Current Value</td>
<td>10869</td>
<td>15441</td>
<td>13070</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>144</td>
<td>3332</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.80</td>
<td>127.98</td>
<td>1.20</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.420</td>
<td>1.202</td>
</tr>
<tr>
<td>Current Value</td>
<td>10869</td>
<td>16010</td>
<td>13977</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>151</td>
<td>1684</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.84</td>
<td>135.58</td>
<td>1.21</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.472</td>
<td>1.285</td>
</tr>
<tr>
<td>Current Value</td>
<td>10898</td>
<td>16044</td>
<td>14814</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>148</td>
<td>2393</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.78</td>
<td>105.15</td>
<td>1.22</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.472</td>
<td>1.359</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>10081</td>
<td>13981</td>
<td>12688</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>143</td>
<td>2868</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.80</td>
<td>93.69</td>
<td>1.20</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.386</td>
<td>1.258</td>
</tr>
<tr>
<td>Current Value</td>
<td>10081</td>
<td>14268</td>
<td>13416</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>149</td>
<td>1101</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.84</td>
<td>25.48</td>
<td>1.21</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.415</td>
<td>1.330</td>
</tr>
<tr>
<td>Current Value</td>
<td>10110</td>
<td>14687</td>
<td>14072</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>147</td>
<td>2971</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.86</td>
<td>132.08</td>
<td>1.22</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.00</td>
<td>1.452</td>
<td>1.391</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>78353</td>
<td>97496</td>
<td>99336</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>427</td>
<td>1039</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.22</td>
<td>312.01</td>
<td>3.14</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.244</td>
<td>1.267</td>
</tr>
<tr>
<td>Current Value</td>
<td>78367</td>
<td>98608</td>
<td>100086</td>
</tr>
<tr>
<td>75% Iterations</td>
<td>435</td>
<td>663</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.78</td>
<td>325.9</td>
<td>3.15</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.258</td>
<td>1.277</td>
</tr>
<tr>
<td>Current Value</td>
<td>79163</td>
<td>102507</td>
<td>102629</td>
</tr>
<tr>
<td>95% Iterations</td>
<td>438</td>
<td>3625</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.96</td>
<td>423.32</td>
<td>3.16</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.294</td>
<td>1.296</td>
</tr>
</tbody>
</table>

* The Proportion is calculated as Current Value/LP Current Value
<table>
<thead>
<tr>
<th>Percentage</th>
<th>Current Value</th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
<th>EOQ</th>
<th>Silver-Meal</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>65458</td>
<td>82120</td>
<td>82729</td>
<td>80274</td>
<td>1.000</td>
<td>1.254</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.74</td>
<td>322.47</td>
<td>3.15</td>
<td>3.15</td>
<td>1.000</td>
<td>1.285</td>
</tr>
<tr>
<td>75%</td>
<td>65469</td>
<td>84162</td>
<td>83709</td>
<td>81254</td>
<td>1.000</td>
<td>1.285</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>6.24</td>
<td>360.38</td>
<td>3.15</td>
<td>3.15</td>
<td>1.000</td>
<td>1.291</td>
</tr>
<tr>
<td>95%</td>
<td>66105</td>
<td>85396</td>
<td>86543</td>
<td>84088</td>
<td>1.000</td>
<td>1.291</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.96</td>
<td>371.80</td>
<td>3.16</td>
<td>3.16</td>
<td>1.000</td>
<td>1.291</td>
</tr>
</tbody>
</table>

* The Proportion is calculated as Current Value/LP
Current Value
** Calculated for a 20% Reduction in the Original Holding costs

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Current Value</th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
<th>EOQ</th>
<th>Silver-Meal</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>75578</td>
<td>92643</td>
<td>95359</td>
<td>91966</td>
<td>1.000</td>
<td>1.225</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.46</td>
<td>328.91</td>
<td>3.15</td>
<td>3.15</td>
<td>1.000</td>
<td>1.225</td>
</tr>
<tr>
<td>75%</td>
<td>75592</td>
<td>93509</td>
<td>95729</td>
<td>92336</td>
<td>1.000</td>
<td>1.237</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.32</td>
<td>366.46</td>
<td>3.15</td>
<td>3.15</td>
<td>1.000</td>
<td>1.237</td>
</tr>
<tr>
<td>95%</td>
<td>76388</td>
<td>94994</td>
<td>97472</td>
<td>94079</td>
<td>1.000</td>
<td>1.243</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>5.98</td>
<td>356.54</td>
<td>3.16</td>
<td>3.16</td>
<td>1.000</td>
<td>1.243</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
<th>Silver-Meal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>95601</td>
<td>120031</td>
<td>100383</td>
<td>95774</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>676</td>
<td>3294</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>12.66</td>
<td>478.9</td>
<td>1.23</td>
<td>1.23</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.255</td>
<td>1.050</td>
<td>1.001</td>
</tr>
</tbody>
</table>

|                | 96722   | 122075  | 100403             | 96722        |
| 75% Iterations | 729     | 1287    | -                  | -            |
| Util. CPU Time | 14.02   | 621.28  | 1.24               | 1.24         |
| Optimality*    | 1.000   | 1.262   | 1.038              | 1.000~        |

|                | 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 4.6.12. Continued

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.**</th>
<th>IP Sol.**</th>
<th>Heuristic Solution**</th>
<th>Silver-Meal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>81567</td>
<td>104444</td>
<td>83373</td>
<td>81567</td>
</tr>
<tr>
<td>50% Iterations</td>
<td>696</td>
<td>1213</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>13.61</td>
<td>639.54</td>
<td>1.21</td>
<td>1.21</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.280</td>
<td>1.022</td>
<td>1.000~</td>
</tr>
</tbody>
</table>

|                | 82463     | 107717    | 83696                | 82463        |
| 75% Iterations | 698       | 1164      | -                    | -            |
| Util. CPU Time | 13.80     | 401.8     | 1.24                 | 1.24         |
| Optimality*    | 1.000     | 1.306     | 1.014                | 1.000~        |

|                | 84758     | 108854    | 84974                | 84758        |
| 95% Iterations | 733       | 1808      | -                    | -            |
| Util. 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.***</th>
<th>IP Sol.***</th>
<th>Heuristic Solution***</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current Value</strong></td>
<td>90516</td>
<td>110948</td>
<td>97949</td>
</tr>
<tr>
<td><strong>50% Iterations</strong></td>
<td>659</td>
<td>1103</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>13.48</td>
<td>613.94</td>
<td>1.21</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.225</td>
<td>1.082</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>91636</td>
<td>114129</td>
<td>97663</td>
</tr>
<tr>
<td><strong>75% Iterations</strong></td>
<td>704</td>
<td>1530</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>13.80</td>
<td>521.00</td>
<td>1.24</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.245</td>
<td>1.065</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>94506</td>
<td>115501</td>
<td>98225</td>
</tr>
<tr>
<td><strong>95% Iterations</strong></td>
<td>760</td>
<td>2073</td>
<td>-</td>
</tr>
<tr>
<td><strong>Util. CPU Time</strong></td>
<td>15.10</td>
<td>587.96</td>
<td>1.25</td>
</tr>
<tr>
<td><strong>Optimality</strong>*</td>
<td>1.000</td>
<td>1.222</td>
<td>1.039</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td>Current Value</td>
<td>10869</td>
<td>15852</td>
<td>13173</td>
</tr>
<tr>
<td>60% Iterations</td>
<td>150</td>
<td>1459</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>0.82</td>
<td>54.70</td>
<td>1.02</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.458</td>
<td>1.261</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td>Current Value</td>
<td>10869</td>
<td>16396</td>
<td>14277</td>
</tr>
<tr>
<td>80% Iterations</td>
<td>149</td>
<td>1370</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>7.36</td>
<td>51.22</td>
<td>1.02</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.508</td>
<td>1.313</td>
</tr>
</tbody>
</table>

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

### Table 4.6.14. One End Item Problem with Structure Three

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td>Current Value</td>
<td>18787</td>
<td>26469</td>
<td>24313</td>
</tr>
<tr>
<td>60% Iterations</td>
<td>154</td>
<td>843</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>1.02</td>
<td>159.4</td>
<td>0.90</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.408</td>
<td>1.294</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOQ</td>
<td>Silver-Meal</td>
<td></td>
</tr>
<tr>
<td>Current Value</td>
<td>19035</td>
<td>26210</td>
<td>24891</td>
</tr>
<tr>
<td>80% Iterations</td>
<td>149</td>
<td>2935</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>1.06</td>
<td>189.00</td>
<td>0.90</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.376</td>
<td>1.307</td>
</tr>
</tbody>
</table>

* The Proportion is calculated as Current Value/LP Current Value
Table 4.6.15. Three End Item Problem with Structure One

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>73623</td>
<td>92215</td>
<td>88048</td>
</tr>
<tr>
<td>60% Iterations</td>
<td>471</td>
<td>1281</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>7.36</td>
<td>168.60</td>
<td>1.49</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.252</td>
<td>1.195</td>
</tr>
<tr>
<td>Current Value</td>
<td>73625</td>
<td>92777</td>
<td>88841</td>
</tr>
<tr>
<td>80% Iterations</td>
<td>477</td>
<td>2137</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>8.10</td>
<td>218.33</td>
<td>1.49</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.260</td>
<td>1.206</td>
</tr>
</tbody>
</table>

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

Table 4.6.16. Three End Item Problem with Structure Three

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>58564</td>
<td>75675</td>
<td>69747</td>
</tr>
<tr>
<td>60% Iterations</td>
<td>431</td>
<td>689</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>6.46</td>
<td>453.54</td>
<td>0.44</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.292</td>
<td>1.190</td>
</tr>
<tr>
<td>Current Value</td>
<td>59773</td>
<td>76037</td>
<td>69974</td>
</tr>
<tr>
<td>80% Iterations</td>
<td>423</td>
<td>762</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>6.96</td>
<td>674.78</td>
<td>0.45</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.272</td>
<td>1.170</td>
</tr>
</tbody>
</table>

* The Proportion is calculated as Current Value/LP Current Value
### Table 4.6.17. Five End Item Problem with Structure One

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
<th>Silver-Meal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current Value</strong></td>
<td>128948</td>
<td>158884</td>
<td>136895</td>
<td>133134</td>
</tr>
<tr>
<td>60% Iterations</td>
<td>717</td>
<td>1035</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>18.16</td>
<td>305.32</td>
<td>2.30</td>
<td>2.30</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.232</td>
<td>1.061</td>
<td>1.032</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>128948</td>
<td>159982</td>
<td>136771</td>
<td>133009</td>
</tr>
<tr>
<td>80% Iterations</td>
<td>745</td>
<td>2431</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>20.44</td>
<td>410.38</td>
<td>2.30</td>
<td>2.30</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.240</td>
<td>1.060</td>
<td>1.031</td>
</tr>
</tbody>
</table>

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

### Table 4.6.18. Five End Item Problem with Structure Three

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
<th>Silver-Meal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current Value</strong></td>
<td>95645</td>
<td>121630</td>
<td>100167</td>
<td>95645</td>
</tr>
<tr>
<td>60% Iterations</td>
<td>800</td>
<td>1284</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>16.80</td>
<td>723.88</td>
<td>2.31</td>
<td>2.31</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.271</td>
<td>1.047</td>
<td>1.000~</td>
</tr>
<tr>
<td><strong>Current Value</strong></td>
<td>97715</td>
<td>124207</td>
<td>100106</td>
<td>97715</td>
</tr>
<tr>
<td>80% Iterations</td>
<td>702</td>
<td>1341</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Util. CPU Time</td>
<td>16.06</td>
<td>887.08</td>
<td>3.31</td>
<td>3.31</td>
</tr>
<tr>
<td>Optimality*</td>
<td>1.000</td>
<td>1.271</td>
<td>1.024</td>
<td>1.000~</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%+</td>
<td>10869</td>
<td>15446</td>
<td>13076</td>
</tr>
<tr>
<td>25% CPU Time</td>
<td>0.84</td>
<td>97.77</td>
<td>1.02</td>
</tr>
<tr>
<td>Util. Optimality*</td>
<td>1.000</td>
<td>1.421</td>
<td>1.202</td>
</tr>
</tbody>
</table>

Table 4.6.20. Three End Item Problem with Structure Two

<table>
<thead>
<tr>
<th></th>
<th>LP Sol.</th>
<th>IP Sol.</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%+</td>
<td>78353</td>
<td>97496</td>
<td>106574</td>
</tr>
<tr>
<td>25% CPU Time</td>
<td>5.80</td>
<td>286.26</td>
<td>1.49</td>
</tr>
<tr>
<td>Util. Optimality*</td>
<td>1.000</td>
<td>1.244</td>
<td>1.360</td>
</tr>
</tbody>
</table>

* The proportion is calculated as Current Value/LP Current Value
Table 4.6.21. Five End Item Problem with Structure Three

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

* The Proportion is calculated as Current Value/LP Current Value
Graph 4.6.1. One End Item Problem with Structure 1

**Current Value**

- LP
- IP
- H1
- H2

**CPU Time (secs)**

- LP
- (IP)/20
- H1 (or H2)

**Optimality**

- LP
- IP
- H1
- H2
Graph 4.6.2. One End Item Problem with Structure 1
(20% Reduction in the Original Costs)

Current Value

Utilisation %

CPU Time (secs)

Utilisation %

Optimality

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

- Current Value
  - LP
  - IP
  - H1
  - H2

- CPU Time (secs)
  - LP
  - (IP)/20
  - H1 (or H2)

- Optimality
  - LP
  - IP
  - H1
  - H2

Utilisation %

0 50 60 70 80 90 100
Graph 4.6.4. Three End Item Problem with Structure 1

- **Graph 1:** Current Value vs. Utilisation %
- **Graph 2:** CPU Time (secs) vs. Utilisation %
- **Graph 3:** Optimality vs. Utilisation %

Legend:
- LP
- IP
- H1
- H2

Note: The graphs show the performance of different items (LP, IP, H1, H2) under varying utilisation percentages (50 to 100%).
Graph 4.6.5. Three End Item Problem with Structure 1
(20% Reduction in the original costs)

- Current Value
- Utilisation %

- CPU Time (secs)
- Utilisation %

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

1. Current Value
   - LP
   - IP
   - H1
   - H2

2. CPU Time (secs)
   - LP
   - (IP)/20
   - H1 (or H2)

3. Optimality
   - LP
   - IP
   - H1
   - H2

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

- Current Value
- CPU Time (secs)
- Optimality

Utilisation %

LP
IP
H1
H2

(IP)/20
H1 (or H2)
Graph 4.6.9. Five End Item Problem with Structure 1
(40% Reduction in the Original Costs)

- Current Value
  - LP
  - IP
  - H1
  - H2

- CPU Time (secs)
  - LP
  - (IP)/20
  - H1 (or H2)

- Optimality
  - LP
  - IP
  - H1
  - H2
Graph 4.6.10. One End Item Problem with Structure 1

- **Current Value**
  - LP
  - IP
  - H1
  - H2

- **CPU Time (secs)**
  - LP
  - (IP)/20
  - H1 (or H2)

- **Optimality**
  - LP
  - IP
  - H1
  - H2
Graph 4.6.12. Five End Item Problem with Structure 3

- Current Value
  - Current Value Graph with lines and markers for LP, IP, H1, and H2.

- CPU Time (secs)
  - CPU Time Graph with lines and markers for LP, (IP)/20, and H1 (or H2).

- Optimality
  - Optimality Graph with lines and markers for LP, IP, H1, and H2.
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 occurred 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:

\[
\text{Total Cost} = \sum_{k=1}^{N} h_k I_{k+1} + \sum_{k=1}^{N} z_k \delta(x_k) + \sum_{k \in \mathcal{B}(x)} u_k
\]

subject to

\[
\begin{align*}
    s_k - u_k & \quad \text{for } k \in \mathcal{B}(x) \\
    z_k = \begin{cases} 
        s_k + v_k & \quad \text{for } k \in \mathcal{C}(x) \\
        s_k & \quad \text{for } k \in \mathcal{D}(x)
    \end{cases}
\end{align*}
\]

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 suboptimisation 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 \( (C_V \)
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 successful 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 cyclical 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 e.g. 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 $P_{it} = \text{cap}_t$ where $P_{it}$ is the units of production item i in period t, and $\text{cap}_t$ 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 $\text{cap}_t = 79$ was found. So the original schedule will be:

<table>
<thead>
<tr>
<th>Period</th>
<th>Demand</th>
<th>Pit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>79</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>42</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Period</th>
<th>Demand</th>
<th>Pit</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>42</td>
<td>81</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>81</td>
</tr>
<tr>
<td>7</td>
<td>48</td>
<td>41</td>
</tr>
<tr>
<td>8</td>
<td>38</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

\[
TC = \sum_{t=1}^{T} h_i \cdot (T - t + 1) \cdot P_{it} + c_{si} \cdot X_{it}
\]

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 $C_{si}$ 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

$$ t_r = \left\lceil \frac{\sum_{t=1}^{T} d_{it}}{Q_i} + 0.99 \right\rceil$$

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
Pit 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. $S_i < d_{in}$)

$$ Z_r = \sum_{t=1}^{T} \frac{d_{it}}{t_r} $$

which is 118 or 117.

So the schedule will be

Period 1 2 3 4 5 6
Demand 38 43 37 42 33 42
Pit 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
Pit 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 $P_{17}$ is 126.

<table>
<thead>
<tr>
<th>Period</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demand</td>
<td>42</td>
<td>33</td>
<td>42</td>
<td>40</td>
<td>48</td>
<td>38</td>
</tr>
<tr>
<td>$P_{it}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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,

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

or

$$\text{AC} = \frac{cs_i + [(1-1)d_{i1} + (2-1)d_{i2} + ... + (T-1)d_{it}]h_it}{T}$$

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 \( Q_{it} = \sum_{j=1}^{n} d_{ij} \) 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:

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demands</td>
<td>38</td>
<td>43</td>
<td>37</td>
<td>42</td>
<td>33</td>
<td>42</td>
</tr>
<tr>
<td>( P_{it} )</td>
<td>193</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42</td>
</tr>
</tbody>
</table>

Total cost for the original plan is 1000.

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

<table>
<thead>
<tr>
<th>Period</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demand</td>
<td>42</td>
<td>33</td>
<td>42</td>
<td>40</td>
<td>48</td>
<td>38</td>
</tr>
<tr>
<td>( P_{it} )</td>
<td>0</td>
<td>0</td>
<td>168</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Heuristic Solution</th>
<th>Without Rolling Schedule</th>
<th>With Rolling Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOQ</td>
<td>Silver-Meal</td>
<td>EOQ</td>
</tr>
<tr>
<td>50%+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75% Util.</td>
<td>177047</td>
<td>158000</td>
</tr>
<tr>
<td>50% Util.</td>
<td>159790</td>
<td>132834</td>
</tr>
<tr>
<td>75% Util.</td>
<td>177121</td>
<td>150165</td>
</tr>
</tbody>
</table>

Table 5.8.2. One End Item Problem with Structure Two

<table>
<thead>
<tr>
<th>Heuristic Solution</th>
<th>Without Rolling Schedule</th>
<th>With Rolling Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOQ</td>
<td>Silver-Meal</td>
<td>EOQ</td>
</tr>
<tr>
<td>50%+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75% Util.</td>
<td>313810</td>
<td>292097</td>
</tr>
<tr>
<td>50% Util.</td>
<td>285904</td>
<td>250619</td>
</tr>
<tr>
<td>75% Util.</td>
<td>315850</td>
<td>280566</td>
</tr>
</tbody>
</table>
Table 5.8.3. One End Item Problem with Structure Three

<table>
<thead>
<tr>
<th>Heuristic Solution</th>
<th>Without Rolling Schedule</th>
<th>With Rolling Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOQ</td>
<td>Silver-Meal EOQ</td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>50%+</td>
<td>322818</td>
<td>263433</td>
</tr>
<tr>
<td>75%</td>
<td>263433</td>
<td>443883</td>
</tr>
<tr>
<td>Util.</td>
<td>262283</td>
<td>406937</td>
</tr>
<tr>
<td>50%</td>
<td>201299</td>
<td>363994</td>
</tr>
<tr>
<td>Util.</td>
<td>219667</td>
<td>327048</td>
</tr>
<tr>
<td>75%</td>
<td>280650</td>
<td>385354</td>
</tr>
<tr>
<td>Util.</td>
<td>348408</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8.4. Three End Item Problem with Structure One

<table>
<thead>
<tr>
<th>Heuristic Solution</th>
<th>Without Rolling Schedule</th>
<th>With Rolling Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOQ</td>
<td>Silver-Meal EOQ</td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>50%+</td>
<td>1065813</td>
<td>937737</td>
</tr>
<tr>
<td>75%</td>
<td>937737</td>
<td>1330438</td>
</tr>
<tr>
<td>Util.</td>
<td>859223</td>
<td>1286940</td>
</tr>
<tr>
<td>50%</td>
<td>1042958</td>
<td>1376117</td>
</tr>
<tr>
<td>Util.</td>
<td>886784</td>
<td>1309539</td>
</tr>
<tr>
<td>75%</td>
<td>1070519</td>
<td>1396423</td>
</tr>
<tr>
<td>Util.</td>
<td>1329845</td>
<td></td>
</tr>
</tbody>
</table>
### Table 5.8.5. Three End Item Problem with Structure Two

<table>
<thead>
<tr>
<th></th>
<th>Heuristic Solution</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without Rolling</td>
<td>With Rolling</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schedule EOQ</td>
<td>Schedule EOQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Silver-Meal</td>
<td>Silver-Meal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%+ 75% Util.</td>
<td>1442978</td>
<td>1275779</td>
<td>1832941</td>
<td>1750423</td>
</tr>
<tr>
<td>50% Util.</td>
<td>1424898</td>
<td>1184607</td>
<td>1882821</td>
<td>1800479</td>
</tr>
<tr>
<td>75% Util.</td>
<td>1436365</td>
<td>1196073</td>
<td>1892480</td>
<td>1810137</td>
</tr>
</tbody>
</table>

### Table 5.8.6. Three End Item Problem with Structure Three

<table>
<thead>
<tr>
<th></th>
<th>Heuristic Solution</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without Rolling</td>
<td>With Rolling</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schedule EOQ</td>
<td>Schedule EOQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Silver-Meal</td>
<td>Silver-Meal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%+ 75% Util.</td>
<td>1020687</td>
<td>910895</td>
<td>1289637</td>
<td>1203334</td>
</tr>
<tr>
<td>50% Util.</td>
<td>959582</td>
<td>808717</td>
<td>1291359</td>
<td>1203364</td>
</tr>
<tr>
<td>75% Util.</td>
<td>968396</td>
<td>817531</td>
<td>1295945</td>
<td>1207950</td>
</tr>
</tbody>
</table>
### Table 5.8.7. Five End Item Problem with Structure One

<table>
<thead>
<tr>
<th>Heuristic Solution Without Rolling Schedule</th>
<th>With Rolling Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOQ</td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>50%+</td>
<td></td>
</tr>
<tr>
<td>75% Util.</td>
<td></td>
</tr>
<tr>
<td>50% Util.</td>
<td>1149856</td>
</tr>
<tr>
<td>75% Util.</td>
<td>1141588</td>
</tr>
</tbody>
</table>

### Table 5.8.8. Five End Item Problem with Structure Two

<table>
<thead>
<tr>
<th>Heuristic Solution Without Rolling Schedule</th>
<th>With Rolling Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOQ</td>
<td>Silver-Meal</td>
</tr>
<tr>
<td>50%+</td>
<td></td>
</tr>
<tr>
<td>75% Util.</td>
<td></td>
</tr>
<tr>
<td>50% Util.</td>
<td>1062284</td>
</tr>
<tr>
<td>75% Util.</td>
<td></td>
</tr>
</tbody>
</table>
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

\[
C_V = \sqrt{\frac{\sum_{i=1}^{n} \left( d_i - \bar{d} \right)^2}{n-1}}
\]

\[
\bar{d} = \frac{\sum_{i=1}^{n} d_i}{n}
\]

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 necessary 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

\[ \text{LP} \leq \text{IP}^* \leq \text{Heuristic solution} \]
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

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

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 $TC_1$ and $TC_2$ 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:

$$TC_1 = TC_2 \quad S_i \delta_1 (P_{it}) + h_i I_{it} = S_i \delta_2 (P_{it})$$

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

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

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 illustrated 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 similar 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 \( i (i = 1, \ldots, N) \) with all the possible \( q \) period problems \( (q = 1, \ldots, t) \). Calculate the costs for each \( q \) period problem using the shortest path procedure and put them in ascending order for each \( t \)-period problem \( (t \leq T-1) \). These costs are calculated by the following formula:

\[
\min_{i=1}^{N} \sum_{t=1}^{T} \left( c_{s_i} \delta(P_{it}) + h_i I_{it} \right)
\]

The notation is the same as before.

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

\[
\sum_{i=1}^{N} [b_i P_{it}] \leq \text{Cap}_t
\]

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:

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

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.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Successor</th>
<th>Predecessor</th>
<th>Setup Cost</th>
<th>Echelon Holding Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>A(i)</td>
<td>B(i)</td>
<td>$S_i$</td>
<td>$e_i$</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>2, 3</td>
<td>10</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-</td>
<td>20</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
<td>8</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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
\[ e_i = h_i - \sum_{j \in C(i)} h_j \]

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

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

<table>
<thead>
<tr>
<th>Period</th>
<th>Demand</th>
<th>Period</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
<td>39</td>
</tr>
</tbody>
</table>

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 \( q \) (\( q = 1, \ldots, t \)) 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 1 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.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(8)[10]</td>
</tr>
<tr>
<td>2</td>
<td>(8)[20]</td>
</tr>
<tr>
<td>3</td>
<td>(8)[8]</td>
</tr>
<tr>
<td></td>
<td>(24)[38]</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(8)[10]</td>
</tr>
<tr>
<td>2</td>
<td>(8)[20]</td>
</tr>
<tr>
<td>3</td>
<td>(8)[8]</td>
</tr>
<tr>
<td></td>
<td>(24)[38]</td>
</tr>
</tbody>
</table>

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;

\[ W(1,1) = \{ V(1, 4 + 1 - 2), 20, 0 \} = \{8,15,4,20,0\} \]
\[ W(1,2) = \{ V(1, 4 + 2 - 2), 12 \} = \{8,15,4,8,12\}, \text{ etc.} \]

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

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

<table>
<thead>
<tr>
<th>Stage</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>(8,15,4,8,12) [50]</td>
</tr>
<tr>
<td>2</td>
<td>(23,0,12,0,12) [83]</td>
</tr>
<tr>
<td>3</td>
<td>(8,19,0,8,12) [36]</td>
</tr>
<tr>
<td></td>
<td>(39,34,16,16,36) [169]</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1  2  3  4  5</td>
</tr>
<tr>
<td>1</td>
<td>(23, 0, 24, 0, 0) [67.0]</td>
</tr>
<tr>
<td>2</td>
<td>(8, 24, 0, 15, 0) [86.0]</td>
</tr>
<tr>
<td>3</td>
<td>(8, 15, 4, 20, 0) [44.0]</td>
</tr>
<tr>
<td></td>
<td>(39, 39, 28, 35, 0) [197.0]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Setup Cost</th>
<th>Holding Cost</th>
<th>H1</th>
<th>H2</th>
<th>Holding Cost</th>
<th>H1</th>
<th>H2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>127.0</td>
<td>80.3</td>
<td>8</td>
<td>0.1</td>
<td>120.5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>141.0</td>
<td>98.0</td>
<td>8</td>
<td>0.2</td>
<td>129.0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>155.0</td>
<td>115.8</td>
<td>8</td>
<td>0.3</td>
<td>137.5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>169.0</td>
<td>133.5</td>
<td>8</td>
<td>0.4</td>
<td>146.0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>183.0</td>
<td>151.3</td>
<td>8</td>
<td>0.5</td>
<td>154.5</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>197.0</td>
<td>169.0</td>
<td>8</td>
<td>0.6</td>
<td>163.0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>211.0</td>
<td>186.7</td>
<td>8</td>
<td>0.7</td>
<td>171.5</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>225.0</td>
<td>204.5</td>
<td>8</td>
<td>0.8</td>
<td>180.0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>239.0</td>
<td>222.3</td>
<td>8</td>
<td>0.9</td>
<td>188.5</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>253.0</td>
<td>240.0</td>
<td>8</td>
<td>1.0</td>
<td>197.0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>267.0</td>
<td>257.8</td>
<td>8</td>
<td>1.1</td>
<td>205.5</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>281.0</td>
<td>275.5</td>
<td>8</td>
<td>1.2</td>
<td>214.0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>295.0</td>
<td>293.3</td>
<td>8</td>
<td>1.3</td>
<td>222.5</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>309.0</td>
<td>311.0</td>
<td>8</td>
<td>1.4</td>
<td>231.0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>323.0</td>
<td>328.8</td>
<td>8</td>
<td>1.5</td>
<td>239.5</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>337.0</td>
<td>346.5</td>
<td>8</td>
<td>1.6</td>
<td>248.0</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>351.0</td>
<td>364.3</td>
<td>8</td>
<td>1.7</td>
<td>256.5</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>365.0</td>
<td>382.0</td>
<td>8</td>
<td>1.8</td>
<td>265.0</td>
</tr>
</tbody>
</table>

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 work well if:

- setup cost is low
- holding cost is high (Setup cost/Holding cost ≤ 16)

Conversely, the heuristic developed in this thesis appears to work well if;

- setup cost is high
- holding cost is low (Setup cost/Holding cost ≥ 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:

\[
T_i = \frac{T_{i-1}}{1 + g T_{i-1}}
\]

where \( g \) is the constant which is defined as the temperature falls from some initial value of \( T_0 \) to a final value of \( T_f \) in \( f \) iteration. This is calculated using the following formula:

\[
g = \frac{(T_0 - T_f)}{((f - 1)T_0 T_f)}
\]

(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


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 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.


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


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.


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


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,' 


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.


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

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


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.


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.


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
ENDDATA
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

<table>
<thead>
<tr>
<th>Cost Set 1:</th>
<th>Cost Set 2:</th>
<th>Cost Set 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$h_i$</td>
<td>$S_i$</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>500</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>400</td>
</tr>
</tbody>
</table>

All $b_i = 1.0$ for the item made on the work centre.
All $a_{ij} = 1.0$ for all predecessor relations shown below:

```
  5 ——— 4 ——— 3 ——— 2 ——— 1
```
One End Item Bottleneck Problem
Randomly Generated Demand Stream

Low Coefficient of Variation, $C_V = 0.1057$

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>43</td>
<td>36</td>
<td>46</td>
<td>46</td>
<td>42</td>
<td>38</td>
<td>41</td>
<td>44</td>
<td>40</td>
<td>35</td>
<td>46</td>
</tr>
</tbody>
</table>

Medium Coefficient of Variation $C_V = 0.1862$

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>54</td>
<td>47</td>
<td>80</td>
<td>63</td>
<td>45</td>
<td>76</td>
<td>74</td>
<td>55</td>
<td>54</td>
<td>58</td>
<td>76</td>
<td>58</td>
</tr>
</tbody>
</table>

High Coefficient of Variation $C_V = 0.3464$

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>31</td>
<td>58</td>
<td>51</td>
<td>64</td>
<td>42</td>
<td>44</td>
<td>18</td>
<td>48</td>
<td>41</td>
<td>58</td>
<td>32</td>
</tr>
</tbody>
</table>
Three End Item Bottleneck Problem

Randomly Generated Cost Sets

<table>
<thead>
<tr>
<th>i</th>
<th>$h_i$</th>
<th>$s_i$</th>
<th>$h_i$</th>
<th>$s_i$</th>
<th>$h_i$</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>500</td>
<td>2.0</td>
<td>500</td>
<td>0.5</td>
<td>400</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>400</td>
<td>2.0</td>
<td>300</td>
<td>1.0</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>300</td>
<td>0.1</td>
<td>300</td>
<td>2.0</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>200</td>
<td>2.0</td>
<td>400</td>
<td>0.1</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>400</td>
<td>2.0</td>
<td>500</td>
<td>1.0</td>
<td>500</td>
</tr>
<tr>
<td>6</td>
<td>0.1</td>
<td>500</td>
<td>1.0</td>
<td>400</td>
<td>0.1</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>2.0</td>
<td>300</td>
<td>2.0</td>
<td>400</td>
<td>2.0</td>
<td>500</td>
</tr>
<tr>
<td>8</td>
<td>1.0</td>
<td>300</td>
<td>2.0</td>
<td>200</td>
<td>0.5</td>
<td>500</td>
</tr>
<tr>
<td>9</td>
<td>2.0</td>
<td>300</td>
<td>2.0</td>
<td>500</td>
<td>0.1</td>
<td>300</td>
</tr>
<tr>
<td>10</td>
<td>0.5</td>
<td>500</td>
<td>0.1</td>
<td>300</td>
<td>1.0</td>
<td>300</td>
</tr>
<tr>
<td>11</td>
<td>2.0</td>
<td>400</td>
<td>1.0</td>
<td>200</td>
<td>0.1</td>
<td>400</td>
</tr>
<tr>
<td>12</td>
<td>0.1</td>
<td>400</td>
<td>0.5</td>
<td>300</td>
<td>0.5</td>
<td>500</td>
</tr>
<tr>
<td>13</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>300</td>
<td>1.0</td>
<td>500</td>
</tr>
<tr>
<td>14</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
<td>300</td>
</tr>
<tr>
<td>15</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
<td>300</td>
<td>1.0</td>
<td>400</td>
</tr>
</tbody>
</table>

All $b_i = 1.0$ for the item made on the work centre.
All $a_{ij} = 1.0$ for all predecessor relations shown below:

```
13 -- 10 -- 7 -- 4 -- 1
14 -- 11 -- 8 -- 5 -- 2
15 -- 12 -- 9 -- 6 -- 3
```
Three End Item Bottleneck Problem

Randomly Generated Demand Stream

Low Coefficient of Variation, $C_v = 0.1144$

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

Medium Coefficient of Variation $C_v = 0.1926$

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

High Coefficient of Variation $C_v = 0.3955$

<table>
<thead>
<tr>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>i 1 2 3 4 5 6 7 8 9 10 11 12</td>
</tr>
<tr>
<td>1 00 07 21 01 41 65 47 54 00 63 43 62</td>
</tr>
<tr>
<td>2 00 117 126 93 55 105 08 126 82 45 18 93</td>
</tr>
<tr>
<td>3 25 00 14 17 29 56 105 99 96 42 58 00</td>
</tr>
</tbody>
</table>
### Five End Item Bottleneck Problem

**Randomly Generated Cost Sets**

<table>
<thead>
<tr>
<th>i</th>
<th>h_i</th>
<th>S_i</th>
<th>Cost Set 1:</th>
<th>h_i</th>
<th>S_i</th>
<th>Cost Set 2:</th>
<th>h_i</th>
<th>S_i</th>
<th>Cost Set 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>200</td>
<td>1.0</td>
<td>200</td>
<td>0.1</td>
<td>200</td>
<td>1.0</td>
<td>200</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>500</td>
<td>0.5</td>
<td>300</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>300</td>
<td>0.1</td>
<td>200</td>
<td>2.0</td>
<td>400</td>
<td>2.0</td>
<td>400</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
<td>300</td>
<td>0.5</td>
<td>300</td>
<td>0.5</td>
<td>300</td>
<td>2.0</td>
<td>400</td>
<td>2.0</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>300</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
<td>200</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>6</td>
<td>2.0</td>
<td>300</td>
<td>2.0</td>
<td>500</td>
<td>2.0</td>
<td>500</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
<td>300</td>
<td>0.5</td>
<td>200</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
<td>500</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>1.0</td>
<td>500</td>
<td>2.0</td>
<td>300</td>
<td>0.1</td>
<td>200</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>9</td>
<td>1.0</td>
<td>500</td>
<td>0.1</td>
<td>300</td>
<td>0.5</td>
<td>500</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>10</td>
<td>2.0</td>
<td>200</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>11</td>
<td>0.1</td>
<td>400</td>
<td>2.0</td>
<td>500</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
</tr>
<tr>
<td>12</td>
<td>0.1</td>
<td>400</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
</tr>
<tr>
<td>13</td>
<td>0.1</td>
<td>300</td>
<td>2.0</td>
<td>200</td>
<td>0.1</td>
<td>200</td>
<td>0.1</td>
<td>500</td>
<td>0.1</td>
</tr>
<tr>
<td>14</td>
<td>2.0</td>
<td>300</td>
<td>0.1</td>
<td>500</td>
<td>2.0</td>
<td>300</td>
<td>0.1</td>
<td>500</td>
<td>2.0</td>
</tr>
<tr>
<td>15</td>
<td>0.1</td>
<td>500</td>
<td>0.1</td>
<td>500</td>
<td>0.5</td>
<td>400</td>
<td>0.1</td>
<td>400</td>
<td>0.5</td>
</tr>
<tr>
<td>16</td>
<td>1.0</td>
<td>400</td>
<td>2.0</td>
<td>300</td>
<td>0.5</td>
<td>200</td>
<td>0.5</td>
<td>200</td>
<td>0.5</td>
</tr>
<tr>
<td>17</td>
<td>2.0</td>
<td>500</td>
<td>0.5</td>
<td>500</td>
<td>0.1</td>
<td>300</td>
<td>0.5</td>
<td>200</td>
<td>0.5</td>
</tr>
<tr>
<td>18</td>
<td>2.0</td>
<td>500</td>
<td>0.1</td>
<td>500</td>
<td>2.0</td>
<td>400</td>
<td>2.0</td>
<td>400</td>
<td>2.0</td>
</tr>
<tr>
<td>19</td>
<td>0.1</td>
<td>500</td>
<td>2.0</td>
<td>200</td>
<td>0.5</td>
<td>200</td>
<td>0.5</td>
<td>200</td>
<td>0.5</td>
</tr>
<tr>
<td>20</td>
<td>1.0</td>
<td>300</td>
<td>0.1</td>
<td>300</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>21</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
</tr>
<tr>
<td>22</td>
<td>1.0</td>
<td>300</td>
<td>1.0</td>
<td>300</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
</tr>
<tr>
<td>23</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
</tr>
<tr>
<td>24</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
</tr>
<tr>
<td>25</td>
<td>1.0</td>
<td>400</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>500</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
</tr>
</tbody>
</table>

All $h_i = 1.0$ for the item made on the work centre.
All $a_{ij} = 1.0$ for all predecessor relations shown below:
### Five End Item Bottleneck Problem

#### Randomly Generated Demand Stream

#### Low Coefficient of Variation, $C_V = 0.0790$

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

#### Medium Coefficient of Variation $C_V = 0.2125$

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

#### High Coefficient of Variation $C_V = 0.4282$

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>15</td>
<td>13</td>
<td>92</td>
<td>41</td>
<td>10</td>
<td>31</td>
<td>87</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>76</td>
<td>15</td>
<td>50</td>
<td>36</td>
<td>00</td>
<td>19</td>
<td>24</td>
<td>00</td>
<td>40</td>
<td>92</td>
<td>83</td>
<td>114</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>07</td>
<td>21</td>
<td>97</td>
<td>87</td>
<td>87</td>
<td>00</td>
<td>82</td>
<td>105</td>
<td>00</td>
<td>79</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>00</td>
<td>02</td>
<td>93</td>
<td>22</td>
<td>78</td>
<td>29</td>
<td>00</td>
<td>00</td>
<td>118</td>
<td>109</td>
<td>02</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>79</td>
<td>62</td>
<td>46</td>
<td>00</td>
<td>10</td>
<td>36</td>
<td>00</td>
<td>18</td>
</tr>
</tbody>
</table>
APPENDIX C
Heuristic Program on Fortran 77
This program calculates the total cost for multi
level lot-sizing problem with bottleneck(s)

program bottleneck

integer t, blkstart, blkend
character*14 filename, namel, cname2

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), scost(25,5,96)
5, prod1(25,5,96), stock4(25,5,96), prod2(25,5,96)
6, sumsil(25)

data (iutd(i), i=1,25)/25*0/

print*, 'blksize=', blkjump=', periods='
read*, blksize, blkjump, periods
print*, 'give the first sub-program name please'

Now reading the first sub-program name. If the
first sub-program name which is 'namel' is equal
to 'roll', calculation is done using rolling
schedule. Otherwise it is done without rolling
schedule for period 1 to 96.

read*, namel

print*, 'give the second sub-program name please'

now reading the second sub-program name. If the
second sub-program name is 'silver', calculation
is executed using silver meal technique. Otherwise
it is executed using economic order quantity from
period 1 to 96.

read*, cname2
print *, 'give data file name please'

now reading data from files
read *, filename
open (unit=7, file=filename, status='old')
open (unit=9, file="output")
write (9, 169)

'ibottle' is the number of bottleneck, and
'ibp(j)' is the position of the bottleneck, 'd' is
the demand, 'aver' is the total or average demand in
accordance with the index, 'hi' is the holding
cost, 'cs' is the setup cost, 'prod' is the amount
of production, 'stock' is the amount of the
available stock, 'cap' is the capacity which is
used, 'icho' is the number of end items, 'iutd' is
the capacity utilisation, 'kbt' is the number of
bottleneck(s), 'lk' is the number of end item(s) for
five-end item problems, 'k' is the number of
item(s) for one or three end item problems

Now reading the end items, bottleneck and capacity
utilisation.

read *, icho, ibottle
do 1 j=1, ibottle
   write (9, 94)
   read *, ibp(j), cap(j)
continue

NOW STARTING THE CALCULATIONS

BLOCKSTART IS EQUAL TO BLOCK LOOP

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
      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
   continue

155
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
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
111 do 41 lk=i1,i2,icho
112          do 8 ich=1,icho,icstep
113             k=lk+ich-1
114                 if (k .eq . ibp (kbt)) then
115                    iutd (lk) =int (sum/ (cap (kbt) *blksize) +
116                                      1+0.99)
117                    write (9,93) cap (kbt), iutd (lk)
118                    write (9,99) ((aver (i, j), j=1,2), i=1,
119                          licho), sum
116             endif
117          enddo
118       enddo
119 do 40 lk=i1,i2,icho
120         do 8 ich=1,icho,icstep
121           k=lk+ich-1
122              if (k .eq . ibp (kbt)) then
123                 iutd (lk) =int (sum/ (cap (kbt) *blksize) +
124                                   1+0.99)
125                 write (9,93) cap (kbt), iutd (lk)
126                 write (9,99) ((aver (i, j), j=1,2), i=1,
127                      licho), sum
128             endif
129         enddo
130     enddo
40     go to 10
106     continue
111     continue
116     continue
120     continue
130     continue
135
156
'ANAL1' IS THE MAIN PROGRAM FOR THREE END ITEM PROBLEM

   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

'ANAL2' IS THE MAIN PROGRAM FOR FIVE END ITEM PROBLEM

   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)
   format('stock',5(2x,f9.0))
   format('production',5(2x,f9.0))
   endif

   if(cname2.ne.'silver')then

   now calling the subroutine cost for non-end-items  
for non-bottleneck item(s) using the eoq technique

   call cost(kbt,k,prod,stock,aver,d,hi,cs,ich  
1,ibottle,ibp,sumc,t,icho,blkstart,blkend,blksize  
2,sumh,suml,summ,sums,dem,c,h,dema,lk,stock1,stock2  
3,stock3,name1)
   sumt=sumt+sumc
   total=total+sumc

   else

   now calling the subroutine costs for non end-items  
for non-bottleneck item(s) using silver-meal  
technique

   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
STARTS OF SUBROUTINE ANAL
FUNCTIONS: TO CALCULATE THE PRODUCTION AND STOCKS
FOR ONE-END ITEM PROBLEM FOR BOTTLENECK(S)
PARAMETERS: IUTD, D, PROD, STOCK, BLKSTART, BLKEND, LK, K

subroutine anal (iutd, d, prod, stock, aver, blkstart, blkend, ich, lk, k, name1)

integer blkstart, blkend
dimension iutd(25), d(5,96), prod(25,5,96), stock(25,5,96), aver(5,3)

idif=0
averl=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((averl-idif).gt.iutd(lk))then
        prod(lk,1,j)=iutd(lk)
        stock(lk,1,j)=prod(lk,1,j)-d(1,j)
        idif=idif+prod(lk,1,j)
    else
        prod(lk,1,j)=averl-idif
        stock(lk,1,j)=prod(lk,1,j)-d(1,j)
        idif=idif+prod(lk,1,j)
    endif
SUBROUTINE ANAL1
FUNCTIONS: TO CALCULATE THE PRODUCTION AND STOCKS
FOR THREE-END END ITEMS PROBLEM FOR BOTTLENECK(S)
PARAMETERS: IUTD, D, PROD, STOCK, LK, K, PERIODS

subroutine anall(iutd, d, prod, stock, aver, blkstart
1, blkend, ich, lk, k, nel)
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.l.or.namel.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(l, 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)-
1+prod(lk, 3, j)
    stock(lk, 1, j)=prod(lk, 1, j)-d(1, j)
  elseif(iprod.lt.iutd(lk)) then
    prod(lk, 3, j)=aver3-idif3
    stock(lk, 3, j)=prod(lk, 3, j)-d(3, j)
    prod(lk, 2, j)=aver2-idif2
    stock(lk, 2, j)=prod(lk, 2, j)-d(2, j)
    prod(lk, 1, j)=aver1-idif1
1+stock (lk, 1, j-1)
    else if (iprodl .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)
    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
    idprod = aver1-idif1+aver3-idif3
    if (idprod .le. 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)
    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 idprod = aver1-idif1-prod (lk, 2, j) - prod (lk, 3, j)
    if (idprod .ge. iutd(lk)) then
        prod (lk, 1, j) = iutd(lk) - prod (lk, 2, j) -
        prod (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
\[ \text{idif1} = \text{idif1} + \text{prod}(1k, 1, j) \]
\[ \text{go to 6} \]
\[ \text{stock}(1k, 3, j) = \text{stock}(1k, 3, j-1) - d(3, j) \]
\[ \text{prod}(1k, 3, j) = 0.0 \]
\[ \text{iprod} = \text{aver1-idif1} + \text{aver2-idif2} \]
\[ \text{iprod1} = \text{aver1-idif1} + d(2, j) - \text{stock}(1k, 2, j-1) \]
\[ \text{if} (\text{iprod} \gt \text{iutd}(1k) \text{ and } \text{iprod1} \gt \text{iutd}(1k)) \text{ then} \]
\[ \text{prod}(1k, 2, j) = d(2, j) - \text{stock}(1k, 2, j-1) \]
\[ \text{stock}(1k, 2, j) = 0.0 \]
\[ \text{prod}(1k, 1, j) = \text{iutd}(1k) - \text{prod}(1k, 2, j) \]
\[ \text{stock}(1k, 1, j) = \text{prod}(1k, 1, j) - d(1, j) \]
\[ 1 + \text{stock}(1k, 1, j-1) \]
\[ \text{elseif} (\text{iprod} \leq \text{iutd}(1k)) \text{ then} \]
\[ \text{prod}(1k, 2, j) = \text{aver2-idif2} \]
\[ \text{stock}(1k, 2, j) = \text{prod}(1k, 2, j) - d(2, j) \]
\[ 1 + \text{stock}(1k, 2, j-1) \]
\[ \text{prod}(1k, 1, j) = \text{aver1-idif1} \]
\[ \text{stock}(1k, 1, j) = \text{prod}(1k, 1, j) - d(1, j) \]
\[ 1 + \text{stock}(1k, 1, j-1) \]
\[ \text{endif} \]
\[ \text{idif2} = \text{idif2} + \text{prod}(1k, 2, j) \]
\[ \text{idif1} = \text{idif1} + \text{prod}(1k, 1, j) \]
\[ \text{go to 6} \]
\[ \text{stock}(1k, 1, j) = \text{stock}(1k, 1, j-1) - d(1, j) \]
\[ \text{prod}(1k, 1, j) = 0.0 \]
\[ \text{if} (\text{stock}(1k, 2, j-1) \geq d(2, j)) \text{ go to 16} \]
\[ \text{if} (\text{stock}(1k, 3, j-1) \geq d(3, j)) \text{ go to 14} \]
\[ \text{iprod} = \text{aver2-idif2} + \text{aver3-idif3} \]
\[ \text{if} (\text{iprod} \gt \text{iutd}(1k)) \text{ go to 50} \]
\[ \text{if} (\text{(aver3-idif3)} \geq \text{iutd}(1k)) \text{ then} \]
\[ \text{prod}(1k, 3, j) = d(3, j) - \text{stock}(1k, 3, j-1) \]
\[ \text{stock}(1k, 3, j) = \text{stock}(1k, 3, j-1) \]
\[ 1 + \text{prod}(1k, 3, j) - d(3, j) \]
\[ \text{else} \]
\[ \text{prod}(1k, 3, j) = \text{aver3-idif3} \]
\[ \text{stock}(1k, 3, j) = \text{prod}(1k, 3, j) - d(3, j) \]
\[ 1 + \text{stock}(1k, 3, j-1) \]
\[ \text{endif} \]
\[ \text{idif3} = \text{idif3} + \text{prod}(1k, 3, j) \]
\[ \text{go to 15} \]
\[ \text{stock}(1k, 3, j) = \text{stock}(1k, 3, j-1) - d(3, j) \]
\[ \text{prod}(1k, 3, j) = 0.0 \]
\[ \text{if} ((\text{aver2-idif2}) \geq \text{iutd}(1k)) \text{ then} \]
\[ \text{prod}(1k, 2, j) = \text{iutd}(1k) - \text{prod}(1k, 3, j) \]
\[ \text{stock}(1k, 2, j) = \text{prod}(1k, 2, j) - d(2, j) \]
\[ 1 + \text{stock}(1k, 2, j-1) \]
\[ \text{else} \]
\[ \text{prod}(1k, 2, j) = \text{aver2-idif2} \]
\[ \text{stock}(lk, 2, j) = \text{prod}(lk, 2, j) - d(2, j) \]

1\+ stock \( \text{stock}(lk, 2, j-1) \)

\[ \text{idif} = \text{idif} + \text{prod}(lk, 2, j) \]

16

stock \( \text{stock}(lk, 2, j) = \text{stock}(lk, 2, j-1) - d(2, j) \)

prod \( \text{prod}(lk, 2, j) = 0.0 \)

if \( \text{stock}(lk, 3, j-1) \geq d(3, j) \)

\[ \text{idif} = \text{idif} + \text{prod}(lk, 3, j) \]

go to 6

1+ stock \( \text{stock}(lk, 3, j-1) \)

else

\[ \text{prod}(lk, 3, j) = \text{aver}3 - \text{idif} \]

stock \( \text{stock}(lk, 3, j) = \text{prod}(lk, 3, j) - d(3, j) \)

1+ stock \( \text{stock}(lk, 3, j-1) \)

endif

\[ \text{idif} = \text{idif} + \text{prod}(lk, 3, j) \]

50

prod \( \text{prod}(lk, 3, j) = d(3, j) - \text{stock}(lk, 3, j-1) \)

stock \( \text{stock}(lk, 3, j) = \text{prod}(lk, 3, j) - d(3, j) \)

1+ stock \( \text{stock}(lk, 3, j-1) \)

if \((\text{aver}2 - \text{idif}2 + \text{prod}(lk, 3, j)) \geq \text{iutd}(lk)) \)

\[ \text{prod}(lk, 2, j) = \text{iutd}(lk) - \text{prod}(lk, 3, j) \]

stock \( \text{stock}(lk, 2, j) = \text{prod}(lk, 2, j) - d(2, j) \)

1+ stock \( \text{stock}(lk, 2, j-1) \)

else

\[ \text{prod}(lk, 2, j) = \text{aver}2 - \text{idif}2 \]

stock \( \text{stock}(lk, 2, j) = \text{prod}(lk, 2, j) - d(2, j) \)

1+ stock \( \text{stock}(lk, 2, j-1) \)

endif

\[ \text{idif} = \text{idif} + \text{prod}(lk, 2, j) \]

52

prod \( \text{prod}(lk, 2, j) = \text{iutd}(lk) - d(3, j) \)

stock \( \text{stock}(lk, 2, j) = \text{prod}(lk, 2, j) - d(2, j) \)

1+ stock \( \text{stock}(lk, 2, j-1) \)

if \((\text{iprod} = \text{aver}1 - \text{idif}1 + \text{prod}(lk, 2, j) + \text{prod}(lk, 3, j)) \geq \text{iutd}(lk)) \)

\[ \text{prod}(lk, 1, j) = \text{iutd}(lk) - \text{prod}(lk, 3, j) - \text{prod}(lk, 2, j) \]

stock \( \text{stock}(lk, 1, j) = \text{prod}(lk, 1, j) - d(1, j) \)

1+ stock \( \text{stock}(lk, 1, j-1) \)

else

\[ \text{prod}(lk, 1, j) = \text{aver}1 - \text{idif}1 \]

stock \( \text{stock}(lk, 1, j) = \text{prod}(lk, 1, j) - d(1, j) \)

1+ stock \( \text{stock}(lk, 1, j-1) \)

endif

162
START OF SUBROUTINE ANAL2

FUNCTIONS: TO CALCULATE THE PRODUCTION AND STOCKS
FOR FIVE-END ITEMS PROBLEM FOR BOTTLENECK(S)

subroutine anal2 (kbt, iutd, d, prod, stock, aver, blkstart, blkend, dem, lk, k, name1)

integer blkstart, blkend
dimension iutd(25), d(5,96), prod(25,5,96), 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)

continue

dem(1, j)=sum+dem(1, j)

sum=0

do 14 i=4,5

sum=sum+d(i,j)

continue

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)

   163
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)
        endif
    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)
        endif
    go to 6
35 stock(lk, 4, j) = stock(lk, 4, j-1) - dem(4, j)
    prod(lk, 4, j) = 0
6 continue
164
aver(j1,2) = sum
aver(j1,1) = sumt
continue

return
end

C C 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)
10 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), 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
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
  suml=0.0
  do 10 i=n,n+2
    sum=cs(i)+sum
    suml=hi(i)+suml
  continue
  c(n)=sum/3.0
  h(n)=suml/3.0
  i=n+3
  sum=0.0
  suml=0.0
  do 11 j=i,i+1
    sum=sum+cs(j)
    suml=hi(j)+suml
  continue
  c(i)=sum/2.0
104 h(i)=suml/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
continue

NOW CALLING THE COST SUBROUTINE FOR THE BOTTLENECK(S) ITEMS FOR ONE-END ITEM PROBLEM(S)

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,namel)
  jflag=1
  kbt=kbt+1
elseif

NOW CALLING THE COST SUBROUTINE FOR THE BOTTLENECK(S) ITEMS FOR THREE-END ITEM PROBLEM(S)

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

call bottle(kbt,k,prod,hi,cs,ibp,sumc,t,icho
1,blkstart,blkend,sumh,c,h,lk,namel)
  jflag=1
  if (icho.eq.3) kbt=kbt+1
elseif

NOW CALLING THE COST SUBROUTINE FOR THE BOTTLENECK(S) ITEMS FOR FIVE-END ITEM PROBLEM(S).

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,namel)

NOW STARTING THE DEMAND CALCULATION FOR FIVE-END ITEM PROBLEMS.

if (icho.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
  91

166
92       continue
  dema(1, j)=sum+dema(1, j)
  sum=0.0
  do 93 i=4,5
       sum=sum+d(i, j)
  continue
  dema(4, j)=sum+dema(4, j)
  continue
  if(icho.eq.5) then
       eq=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
  int=aver(ich, 1)/intvl

  ALL DEMANDS ARE PRODUCED IN PERIOD 1 TO CALCULATE
  THE COST

  idifl=0
  ico=0
  averl=aver(ich, 1)
  if(icho.eq.5.and.blkstart.eq.1) then
       stockl(lk, ich, blkstart-1)=0.0
  elseif(icho.ne.5.and.blkstart.eq.1) then
       stocks(k, ich, blkstart-1)=0.0
  elseif(icho.eq.5.and.blkstart.gt.1) then
       idifl=idifl+stockl(lk, ich, blkstart-1)
  else
       idifl=idifl+stockl(k, ich, blkstart-1)
       print*, 'stockl(', k, ich, blkstart-
       c
       l, ')=', stockl(k, ich, blkstart-1)
  endif
  do 80 j=blkstart, blkend
  if(icho.eq.5) then
       if(stockl(lk, ich, j-1).ge.dema(ich, j)) go to 81
       ico=ico+1
       itl(ico)=j
       if((averl-idifl).lt.aver(ich, 1)) go to 400
       stockl(lk, ich, j)=aver(ich, 1)+stockl(lk
       1, ich, j-1)-dema(ich, j)
       idifl=averl-stockl(lk, ich, j-1)+idifl
  else
       if(stockl(k, ich, j-1).ge.d(ich, j)) go to 81
       ico=ico+1
       itl(ico)=j
       if((averl-idifl).lt.aver(ich, 1)) go to 400
       stockl(k, ich, j)=aver(ich, 1)+stockl(k
       1, ich, j-1)-d(ich, j)
       idifl=averl-stockl(k, ich, j-1)+idifl
  endif
  go to 80
  400   if(icho.eq.5) then
idifl=averl-idifl
stockl(lk,ich,j)=idifl-dema(ich,j)
l+stockl(lk,ich,j-1)
else
idifl=averl-idifl
stockl(k,ich,j)=idifl-d(ich,j)
l+stockl(k,ich,j-1)
endif
80 continue
81 if (icho . eq . 5) then
    stockl(lk,ich,j)=stockl(lk,ich,j-1)-ldema(ich,j)
else
    stockl(k,ich,j)=stockl(k,ich,j-1)-ld(ich,j)
endif
83 continue
sum=0.0
do 83 i=1, ico
    sum=sum+idifl*(t-itl(i)+1)
do 83
85 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
idifl=0
averl=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
    idifl=idifl+stock2(lk,ich,blkstart-1)
else
    idifl=idifl+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-idifl).lt.eoq) go to 21
if (icho.eq.5) then
    stock2(lk,ich,j)=stock2(lk,ich,j-1)+eoq-l dema(ich,j)
else
stock2(k,ich,j)=stock2(k,ich,j-1)+eoq-
1d(ich,j)
endif
endif
idifl=idifl+eoq
go to 40
21
ordl=aver(ich,1)-idifl
if(icho.eq.5)then
stock2(lk,ich,j)=stock2(lk,ich,j-1)+ordl-
dema(ich,j)
else
stock2(k,ich,j)=stock2(k,ich,j-1)+ordl-
1d(ich,j)
endif
idifl=idifl+ordl
go to 40
20
if(icho.eq.5)then
stock2(lk,ich,j)=stock2(lk,ich,j-1)-
dema(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

EOQ IS USED TO FIND THE COST3

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)
npl=aver(ich,1)-intvl*ord
np0=intvl-npl
iflag=0
idifl=0
averl=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) )
      go 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
  endif
  281 if ((averl-idif1).le.ord) go to 258
    ordl=ord+1
    if (icho.eq.5) then
      stock3(lk,ich,j)=ordl+stock3(lk,ich,j-1)-
      ldema(ich,j)
    else
      stock3(k,ich,j)=ordl+stock3(k,ich,j-1)-
      ld(ich,j)
    endif
    idif1=idif1+ordl
    go to 26
  endif
  28 if ((averl-idif1).le.ord) go to 258
    if (icho.eq.5) then
      stock3(lk,ich,j)=ord+stock3(lk,ich,j-1)-
      ldema(ich,j)
    else
      stock3(k,ich,j)=ord+stock3(k,ich,j-1)-
      ld(ich,j)
    endif
    idif1=idif1+ord
    go to 26
  endif
  258 ord2=averl-idif1
  if (icho.eq.5) then
    stock3(lk,ich,j)=ord2-dema(ich,j)
  else
    stock3(k,ich,j)=ord2-d(ich,j)
  endif
  idif1=idif1+ord2
  go to 26
  27 if (icho.eq.5) then
stock3(lk, ich, j) = stock3(lk, ich, j-1) - ldema(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)
continue
if (npl.lt.1) go to 31
do 30 i=np0+1, npl+np0
  sum=ordl*(t-itl(i)+1)+sum
continue

C
C ORDL IS USED TO FIND THE COST4
C
C
31 if (icho.eq.5) then
  cost4=c(k)*(npl+np0)+h(k)*sum
else
  cost4=cs(k)*(npl+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)
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)
  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 (namel . ne. 'roll') go to 7

NOW STARTING THE COST CALCULATION OF THE THREE PERIOD FOR FUTURE USAGE FOR ROLLING SCHEDULE

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
   averl = 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))
           1 go to 36
       ico = ico + 1
       itl(ico) = j
           if (((averl - idif1) . lt. aver(ich, 1)) go to 263
               stock1(lk, ich, j) = averl - 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))
                   1 go to 36
               ico = ico + 1
               itl(ico) = j
                   if (((aver1 - idif1) . lt. aver(ich, 1)) go to 263
                   1 go to 263

172
stock1(k, ich, j) = averl - idif1
1+stock1(k, ich, j-1) - d(ich, j)

idif1 = averl - stock1(k, ich, j-1) + idif1

endif

263 if(icho.eq.5) then
    stock1(lk, ich, j) = averl - idif1 - dema(ich, j)
    else
    stock1(k, ich, j) = averl - idif1 - d(ich, j)
    endif

idif1 = averl - idif1

35 continue

if(ico.lt.1) go to 45

do 37 i=1, ico
    if(icho.eq.5) then
        suml(k) = c(k) * iaco + h(k) * (t-itl(ico) +1) * IDIF1
    sumn(k) = suml(k)
    sums(k) = suml(k)
    else
        suml(k) = cs(k) * iaco + hi(k) * (t-itl(ico) +1) * IDIF1
    sumn(k) = suml(k)
    sums(k) = suml(k)
    endif
37 continue

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

averl = aver(ich, 1)
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
  idifl=idifl+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))
      go 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((averl-idifl).lt.eoq) go to 55
  if(icho.eq.5) then
    stock2(lk,ich,j)=stock2(lk,ich,j-1)+eoq-
    ldema(ich,j)
  else
    stock2(k,ich,j)=stock2(k,ich,j-1)+eoq-
    ld(ich,j)
  endif
  idifl=idifl+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-
    ldema(ich,j)
  else
    stock2(k,ich,j)=stock2(k,ich,j-1)+ordl-
    ld(ich,j)
  endif
  idifl=idifl+ordl
  go to 9
56
  if(icho.eq.5) then
    stock2(lk,ich,j)=stock2(lk,ich,j-1)-
    ldema(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

174
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
    averl=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) )
          go to 134
        else
          if (stock3(k,ich,j-1) .ge. dich, j) )
            go to 134
        endif
      else
        iflag=1
        ico=lico+1
        if(iflag.eq.1) go to 282
        if(lico.le.np0) go to 135
      endif
135  if ((averl-idif1).lt.ord) go to 267
    ordl=ordl+1
    itl(ico)=j
    if (icho.eq.5) then
      stock3(lk,ich,j)=ordl+stock3(lk,ich,j-11)-dema(ich,j)
    else
      stock3(k,ich,j)=ordl+stock3(k,ich,j-11)-d(ich,j)
    endif
    idif1=idif1+ordl
    go to 133
282  if (((aver1-idif1).lt.ord) go to 267
    ordl=ordl+1
    itl(ico)=j
    if (icho.eq.5) then
      stock3(lk,ich,j)=ordl+stock3(lk,ich,j-11)-dema(ich,j)
    else
      stock3(k,ich,j)=ordl+stock3(k,ich,j-11)-d(ich,j)
    endif
    idif1=idif1+ordl
    go to 133

else
    stock3(k, ich, j) = ord + stock3(k, ich, j - 1) -
    endif

    itl(ico) = j
    idif1 = idif1 + ord
    go to 133

267
    ord2 = averl - 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) -
        ldema(ich, j)
    else
        stock3(k, ich, j) = stock3(k, ich, j - 1) -
        ld(ich, j)
    endif

133
    continue

sum = 0.0
    if (ico .lt. 1) go to 138
    if (ico .eq. l) go to 137
    do 139 i = l, ico - 1
    sum = sum + ord * (t - itl(i) + 1)
    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)


```fortran
suml(k) = suml(k) = suml(k)
sumn(k) = sumn(k) = sumn(k)
endif

138  suml(k) = 0.0
sumn(k) = 0.0
sums(k) = 0.0
7  if (namel .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
  continue
go to 77
107  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)
```

177
START OF SUBROUTINE COSTS

FUNCTIONS: TO CALCULATE THE COSTS FOR NON-BOTTLENECK ITEM(S) USING SILVER MEAL TECHNIQUE

PARAMETERS: KBT, K, PROD, STOCK, AVER, D, HI, CS, IBOTTLE, IBP, SUMC, T, ICHO, BLKSTART, BLKEND, BLKSIZE, SUMH, SUML, SUMN, SUMS, DEM, C, H, DEMA, LK, STOCK1, STOCK2, STOCK3, SCOST, PROD1, STOCK4, PROD2, SUMSIL

SUBROUTINE COSTS(KBT, K, PROD, STOCK, AVER, D, HI, CS, ICH, IBOTTLE, IBP, SUMC, T, ICHO, BLKSTART, BLKEND, SUMH, SUML, DEM, C, H, DEMA, LK, STOCK1, STOCK2, STOCK3, SCOST, PROD1, STOCK4, PROD2, SUMSIL, NAME1)

INTEGER BLKSTART, BLKEND

DIMENSION PROD(25,5,96), STOCK(25,5,96), AVER(5,3), HI(25), CS(25), ITL(96), D(5,96), IBP(25), SUMH(25), DEM(5,96), C(25), H(25), DEMA(5,96), STOCK1(25,5,96), STOCK2(25,5,96), STOCK3(25,5,96), SCOST(25,5,96), PROD1(25,5,96), STOCK4(25,5,96), PROD2(25,5,96), SUMSIL(25)

IF(KBT.GT.1. OR. ICHO.LT.5. OR. K.GT.3) GO TO 45

DO 42 N=1,21,5
  SUM=0.0
  SUML=0.0
  DO 43 I=N,N+2
    SUM=CS(I)+SUM
    SUML=HI(I)+SUML
  CONTINUE
  C(N)=SUM/3.0
  H(N)=SUML/3.0
  I=N+3
  SUM=0.0
  SUML=0.0
  DO 61 J=I,I+1
    SUM=SUM+CS(J)
    SUML=SUML+HI(J)
  CONTINUE
  C(I)=SUM/2.0
  H(I)=SUML/2.0
  CONTINUE
  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
    CONTINUE
  CONTINUE
  C

178
NOW CALLING THE SUBROUTINE BOTTLE FOR THE BOTTLENECK(S) FOR ONE-END ITEM PROBLEM FOR BOTTLENECK(S) ITEM(S)

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,namel)
   jflag=1
   kbt=kbt+1
elseif

NOW CALLING THE SUBROUTINE BOTTLE FOR THE BOTTLENECK(S) FOR THREE-END ITEM PROBLEM(S) FOR BOTTLENECK(S) ITEM(S)

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,namel)
   jflag=1
   if(ich.eq.3)kbt=kbt+1
elseif

NOW CALLING THE SUBROUTINE BOTTLE FOR THE BOTTLENECK(S) FOR FIVE-END ITEM PROBLEM(S) FOR BOTTLENECK(S) ITEM(S)

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,namel)
   if(ich.eq.4)kbt=kbt+1
   jflag=1
endif

NOW STARTING THE DEMAND CALCULATION FOR FIVE-END ITEM PROBLEM

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)
      dema(1,j)=sum+dema(1,j)
91
92
sum=0.0
  do 93 i=4,5
    sum=sum+d(i,j)
93    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))
        lgo to 30
      prodl(lk,ich,j)=prodl(lk,ich,j-1)
      prod2(lk,ich,blkstart+ik)=prodl(lk,ich,1,j)
      1+dema(ich,j)
5else
        if (stock4(k,ich,j-1).ge.d(ich,j))
6go 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
        prodl(k,ich,j)=prodl(k,ich,j-1)
        prod2(k,ich,blkstart+ik)=prodl(k,ich,1,ich,j)
      1+d(ich,j)
      endif
      2go to 20
24  if (icho.eq.5) then
    stock4(lk,ich,j)=stock4(lk,ich,j-1)-
    ldema(ich,j)
    it=it+1
    ik=ik+1
    ic=0
    sum=cs(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
  go to 20
  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) )
         igo 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,ich,it-1) )
goto 50
      1
      prod1(lk,ich,m-1)=prod1(lk,ich,m-1)+dema(ich,m)
      prod2(lk,ich,m-1)=prod2(lk,ich,m-1)
      else
      if(stock4(k,ich,m-1) .ge. d(ich,m) )
goto 27
      it=it+1
      ic=ic+1
      sum=sum+(ic-1)*h(k)*d(ich,m)
      scost(k,ich,it)=sum/ic
      if(scost(k,ich,it) .gt. scost(k,ich,it-1) )
goto 50
      1
      prod1(k,ich,m-1)=prod1(k,ich,m-1)+d(ich,m)
      prod2(k,ich,m-1)=prod2(k,ich,m-1)
      endif
   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)
      181
\begin{verbatim}
scost(1k, 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

50  if (m. gt . blkend) go to 51
  l = 0
  it = blkstart+ik
  ic = 1
  if (icho . eq. 5) then
    prod1(1k, ich, m) = dema(ich, m)
    prod2(1k, ich, m) = prod1(1k, ich, m)
    sum = c(k)
    scost(1k, 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)
     endif
   go to 46
47  if (prod2(1k, ich, j) . gt . 0.0) then
     sum = sum + hi(k) * (t-j+1) * prod2(1k, ich, j)
     ico = ico + 1
     stock4(1k, ich, j) = prod2(1k, ich, j) + stock4(1k, 1, ich, j-1) - dema(ich, j)
   else
     stock4(1k, ich, j) = prod2(1k, ich, j) + stock4(1k, 1, ich, j-1) - dema(ich, j)
   endif
46  continue
     if (icho . eq. 5) then
       sumy = c(k) * ico + sum
\end{verbatim}
else
    sumy=cs(k)*ico+sum
endif
if (namel 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
icol=0
do 31 j=blkstart, blkstart+2
    if (icho.eq.5) then
        if (prod2(1k, ich, j) .le. 0.0) go to 32
        icol=icol+1
        ssil=ssil+h(k)*(t-j+1)*prod2(1k, ich, j)
    else
        if (prod2(k, ich, j) .le. 0.0) go to 32
        icol=icol+1
        ssil=ssil+hi(k)*(t-j+1)*prod2(k, ich, j)
    endif
32 continue
if (icho.eq.5) then
    sumsil(k)=icol*c(k)+ssil
else
    sumsil(k)=icol*cs(k)+ssil
endif
44 write (9,103) k, sumy, sumc, sumsil(k)
103 format (i4,5x, 3f13.2)
19 return
end

C

C START OF SUBROUTINE BOTTLE
C FUNCTIONS: TO CALCULATE THE COSTS FOR BOTTLENECK ITEM(S)
C PARAMETERS: KBT, K, PROD, HI, CS, IBP, SUMC, T, ICHO, SUMH, C, H, LK, PERIODS

subroutine bottle(kbt, k, prod, hi, cs, ibp, sumc, t, icho, blkstart, blkend, sumh, c, h, lk, namel)
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
  costl=sum+ip*c(k)
else
  costl=sum+ip*cs(k)
endif
if(namel.ne.'roll') then
  sumh(k)=0.0
  sumc=costl
  go to 6
endif
if(blkstart.eq.1) then
  sumhold=0.0
else
  sumhold=sumh(k)
endif
sumc=costl+sumhold
ip=0
do 20 j=blkstart,blkstart+2
  if(icho.eq.5) goto 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
write(9,10) k, cost1, sumc, sumh(k)
format(' position of bottleneck =', i4, f13.1
1, f8.0, f8.0)
return
end
APPENDIX D
NAG Subroutine Program
This program is to show the uniform distribution

real*8 x
integer i
double precision g05caf
external g05caf
external g05cbf
external g05ccf
write (6,120)
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 results/1x)
121 format(1x,f10.4)
stop
end
APPENDIX E
Reprint of Published Papers
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.

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

Revision received June 1991.
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.

![Diagram](image)

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 \( i \).
- \( c_{pi} \): available capacity of the work centre at time \( t \).
- \( c_{si} \): 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 \( i \).
- \( I_i \): 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:

\[
Z = \sum_{i=1}^{N} \sum_{t=1}^{T} \left[ h_i (T-t+1) P_{it} + c_{si} X_{it} \right]
\]

Subject to:

\[
\sum_{n=1}^{i} \left[ P_{i,n-L_i} - \sum_{j=1}^{i} a_{ij} P_{jm} \right] \geq \sum_{n=1}^{i} d_{in} - I_{i0} \quad i = 1, \ldots, N \quad t = 1, \ldots, T
\]

\[
\sum_{i=1}^{N} [b_i P_{it} + s_i X_{it}] \leq c_{pi} \quad t = 1, \ldots, T
\]

\[
X_{it} = \begin{cases} 
1 & \text{if } P_{it} > 0 \\
0 & \text{otherwise}
\end{cases} \quad i = 1, \ldots, N \quad t = 1, \ldots, T
\]

\[
P_{it} \geq 0 \quad i = 1, \ldots, N \quad t = 1, \ldots, T
\]

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 \( c_s \) 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 \( S_{it} < d_{in} \)).

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

\[
t_i = \left\lceil \frac{\sum_{i=1}^{T} d_{in}/Q_i + 0.5}{Q_i} \right\rceil
\]

and production is made in any period \( n \) whenever \( S_{in} < d_{in} \). Note that \( \lceil \cdot \rceil \) is the integer part function.

If in any period \( n \)

\[
Q_i \geq \sum_{i=n}^{T} d_{in} - S_{in}
\]

then set

\[
Q_i = \sum_{i=n}^{T} d_{in} - S_{in}
\]
(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 $S_{in} < d_{in}$

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

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.

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 $\text{cap}_i$ will allow in period 1, i.e. set $P_{i1} = \text{cap}_i$, 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.

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

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

Case (b). 3-end-item problems

A three period cycle is adopted.

Period 1 Set $P_{21} = d_{21}$, $P_{31} = d_{31}$ and $P_{11} = \text{cap}_1 - d_{21} - d_{31}$.

Period 2 Set $P_{32} = d_{32}$, $P_{12} = 0$ and $P_{22} = \text{cap}_2 - d_{32}$ provided $S_{12} > d_{12}$.

Otherwise set $P_{12} = d_{12} + d_{13} - S_{12}$, $P_{32} = d_{32}$ and $P_{22} = \text{cap}_2 - P_{12} - P_{32}$.

Period 3 Set $P_{33} = \text{cap}_3$ 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 $P_{33} = \text{cap}_3 - P_{13} - P_{23}$.

Period 4 Set $P_{14} = \text{cap}_4$ provided $S_{24} > d_{24}$ and $S_{34} > d_{34}$.

Period 5 Set $P_{25} = \text{cap}_5$ provided $S_{15} > d_{15}$ and $S_{35} > d_{35}$.

Period 6 Set $P_{36} = \text{cap}_6$ 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
Multi-level lot-sizing problems with bottlenecks

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Current value</th>
<th>LP solution</th>
<th>IP solution</th>
<th>Heuristic solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>10869</td>
<td>15511</td>
<td>13072</td>
<td></td>
</tr>
<tr>
<td></td>
<td>158</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.03</td>
<td>56.36</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.70</td>
<td>0.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>10869</td>
<td>16610</td>
<td>13982</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>373</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.10</td>
<td>59.17</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>0.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td>10899</td>
<td>17314</td>
<td>14827</td>
<td></td>
</tr>
<tr>
<td></td>
<td>178</td>
<td>260</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.19</td>
<td>69.24</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.63</td>
<td>0.74</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*The percentage of LP solution.

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

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Current value</th>
<th>LP solution</th>
<th>IP solution</th>
<th>Heuristic solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>30036</td>
<td>37085</td>
<td>35789</td>
<td></td>
</tr>
<tr>
<td></td>
<td>175</td>
<td>757</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.15</td>
<td>52.86</td>
<td>3.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.81</td>
<td>0.84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>30036</td>
<td>38373</td>
<td>37373</td>
<td></td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>1019</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>65.04</td>
<td>3.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.78</td>
<td>0.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td>30031</td>
<td>39035</td>
<td>38720</td>
<td></td>
</tr>
<tr>
<td></td>
<td>171</td>
<td>572</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.20</td>
<td>39.68</td>
<td>3.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.77</td>
<td>0.74</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*The percentage of LP solution.

Table 2. One-end-item problem with cost structure 2.
<table>
<thead>
<tr>
<th>Utilization</th>
<th>Current Value</th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>18787</td>
<td>24777</td>
<td>23716</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>171</td>
<td>872</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CPU time</td>
<td>1:17</td>
<td>47.57</td>
<td>4:14</td>
</tr>
<tr>
<td></td>
<td>Optimality*</td>
<td>-</td>
<td>0:76</td>
<td>0:79</td>
</tr>
<tr>
<td>75%</td>
<td>18863</td>
<td>26770</td>
<td>24596</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>167</td>
<td>1182</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CPU time</td>
<td>1:18</td>
<td>58.24</td>
<td>4:14</td>
</tr>
<tr>
<td></td>
<td>Optimality*</td>
<td>-</td>
<td>0:70</td>
<td>0:77</td>
</tr>
<tr>
<td>95%</td>
<td>19375</td>
<td>27684</td>
<td>25505</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>168</td>
<td>365</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CPU time</td>
<td>1:12</td>
<td>55.04</td>
<td>4:14</td>
</tr>
<tr>
<td></td>
<td>Optimality*</td>
<td>-</td>
<td>0:70</td>
<td>0:76</td>
</tr>
</tbody>
</table>

* The percentage of LP solution.

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

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Current Value</th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>73623</td>
<td>91291</td>
<td>88097</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>519</td>
<td>1194</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CPU time</td>
<td>6:78</td>
<td>180:13</td>
<td>5:04</td>
</tr>
<tr>
<td></td>
<td>Optimality*</td>
<td>-</td>
<td>0:81</td>
<td>0:84</td>
</tr>
<tr>
<td>75%</td>
<td>73623</td>
<td>93900</td>
<td>87991</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>517</td>
<td>860</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CPU time</td>
<td>7:15</td>
<td>217:39</td>
<td>5:04</td>
</tr>
<tr>
<td></td>
<td>Optimality*</td>
<td>-</td>
<td>0:78</td>
<td>0:84</td>
</tr>
<tr>
<td>95%</td>
<td>73951</td>
<td>95611</td>
<td>91925</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>511</td>
<td>869</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CPU time</td>
<td>7:29</td>
<td>206:22</td>
<td>5:04</td>
</tr>
<tr>
<td></td>
<td>Optimality*</td>
<td>-</td>
<td>0:77</td>
<td>0:80</td>
</tr>
</tbody>
</table>

* The percentage of LP solution.

Table 4. Three-end-item problem with cost structure 1.
Multi-level lot-sizing problems with bottlenecks

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

<table>
<thead>
<tr>
<th></th>
<th>LP solution</th>
<th>IP solution</th>
<th>Heuristic solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>50% utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>78962</td>
<td>98882</td>
<td>101095</td>
</tr>
<tr>
<td>Iterations</td>
<td>507</td>
<td>711</td>
<td>—</td>
</tr>
<tr>
<td>CPU time</td>
<td>6.89</td>
<td>143.84</td>
<td>6.11</td>
</tr>
<tr>
<td>Optimality*</td>
<td>—</td>
<td>0.79</td>
<td>0.78</td>
</tr>
<tr>
<td><strong>75% utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>78976</td>
<td>99501</td>
<td>101217</td>
</tr>
<tr>
<td>Iterations</td>
<td>480</td>
<td>1196</td>
<td>—</td>
</tr>
<tr>
<td>CPU time</td>
<td>6.83</td>
<td>155.70</td>
<td>6.11</td>
</tr>
<tr>
<td>Optimality</td>
<td>—</td>
<td>0.79</td>
<td>0.78</td>
</tr>
<tr>
<td><strong>95% utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>79772</td>
<td>102966</td>
<td>102877</td>
</tr>
<tr>
<td>Iterations</td>
<td>507</td>
<td>1349</td>
<td>—</td>
</tr>
<tr>
<td>CPU time</td>
<td>7.80</td>
<td>186.75</td>
<td>6.11</td>
</tr>
<tr>
<td>Optimality</td>
<td>—</td>
<td>0.77</td>
<td>0.78</td>
</tr>
</tbody>
</table>

* The percentage of LP solution.

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

<table>
<thead>
<tr>
<th></th>
<th>LP solution</th>
<th>IP solution</th>
<th>Heuristic solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>50% utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>58424</td>
<td>75418</td>
<td>72346</td>
</tr>
<tr>
<td>Iterations</td>
<td>514</td>
<td>1136</td>
<td>—</td>
</tr>
<tr>
<td>CPU time</td>
<td>6.61</td>
<td>181.64</td>
<td>6.54</td>
</tr>
<tr>
<td>Optimality*</td>
<td>—</td>
<td>0.77</td>
<td>0.81</td>
</tr>
<tr>
<td><strong>75% utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>59061</td>
<td>72219</td>
<td>71670</td>
</tr>
<tr>
<td>Iterations</td>
<td>487</td>
<td>1042</td>
<td>—</td>
</tr>
<tr>
<td>CPU time</td>
<td>6.28</td>
<td>174.74</td>
<td>6.54</td>
</tr>
<tr>
<td>Optimality</td>
<td>—</td>
<td>0.81</td>
<td>0.82</td>
</tr>
<tr>
<td><strong>95% utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>62103</td>
<td>78455</td>
<td>71815</td>
</tr>
<tr>
<td>Iterations</td>
<td>497</td>
<td>924</td>
<td>—</td>
</tr>
<tr>
<td>CPU time</td>
<td>6.49</td>
<td>183.98</td>
<td>6.54</td>
</tr>
<tr>
<td>Optimality</td>
<td>—</td>
<td>0.79</td>
<td>0.86</td>
</tr>
</tbody>
</table>

* The percentage of LP solution.
<table>
<thead>
<tr>
<th>Utilization</th>
<th>Current Value</th>
<th>Iterations</th>
<th>CPU Time</th>
<th>Optimality*</th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>128948</td>
<td>997</td>
<td>2006</td>
<td>—</td>
<td>159486</td>
<td>32818</td>
<td>138185</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>128948</td>
<td>932</td>
<td>1992</td>
<td>0.79</td>
<td>163464</td>
<td>39608</td>
<td>137706</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td>129030</td>
<td>949</td>
<td>2061</td>
<td>0.74</td>
<td>175342</td>
<td>35686</td>
<td>137612</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

* The percentage of LP solution.

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

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Current Value</th>
<th>Iterations</th>
<th>CPU Time</th>
<th>Optimality*</th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>107807</td>
<td>952</td>
<td>1873</td>
<td>-</td>
<td>131818</td>
<td>39675</td>
<td>108674</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>107856</td>
<td>943</td>
<td>1930</td>
<td>0.82</td>
<td>135130</td>
<td>39721</td>
<td>109484</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td>108703</td>
<td>912</td>
<td>1929</td>
<td>0.76</td>
<td>143170</td>
<td>41676</td>
<td>112093</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

* The percentage of LP solution.

Table 8. Five-end-item problem with cost structure 2.
Multi-level lot-sizing problems with bottlenecks

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

* 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...
Multi-level lot-sizing problems with bottlenecks

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

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

THIS PAPER IS CIRCULATED FOR DISCUSSION PURPOSES AND ITS CONTENTS SHOULD BE CONSIDERED PRELIMINARY AND CONFIDENTIAL. NO REFERENCE TO MATERIAL CONTAINED HEREIN MAY BE MADE WITHOUT THE CONSENT OF THE AUTHORS.
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 successful 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 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 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 \( i \),

\( \text{cap}_t \) = Available capacity of the work centre at time \( t \),

\( \text{cs}_i \) = 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,

\[ T = \text{Total number of periods}, \]
\[ M = \text{The number of bottlenecks}, \]
\[ X_{it} = \text{Production indicator; equal to 1 if } P_{it} > 0 \text{ and }
zero otherwise. \]

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:

\[
Z = \sum_{i=1}^{N} \sum_{t=1}^{T} \left[ h_{i} \left( T-t+1 \right) P_{it} + c_{si} X_{it} \right]
\]

Subject to:

\[
\sum_{n=1}^{t} \left( P_{i,n-li} - \sum_{j=1}^{t} a_{ij} P_{jn} \right) \geq \sum_{n=1}^{t} d_{in} - d_{lo} \quad i=1, \ldots, N \quad t=1, \ldots, T
\]  

\[
\sum_{i=1}^{N} \left[ b_{il} P_{it} + s_{il} X_{it} \right] \leq c_{apt} \quad t=1, \ldots, T
\]

\[ X_{it} = \begin{cases} 
1 & \text{if } P_{it} > 0 \\
0 & \text{otherwise} 
\end{cases} \quad i=1, \ldots, N \quad t=1, \ldots, T
\]

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 $t$

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

$$t_i = \left\lfloor \frac{\sum_{n=1}^{t-1} d_{nt}}{Q_i} + 0.5 \right\rfloor$$

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

Note that $\left\lfloor \right\rfloor$ is the integer part function.
If in any period \( n \)
\[
Q_i \geq \sum_{t=n}^{T} d_{it} - S_{tn} \quad \text{then set} \quad Q_i = \sum_{t=n}^{T} d_{it} - S_{tn} .
\]

(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 \( S_{in} < d_{in} \)).

\[
Z_i = (\sum_{t=1}^{T} d_{it}) / t_i
\]

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 3rd 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 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 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 produces as much of non-end-item i as $\text{cap}_t$ will allow in period 1 i.e. set $P_{i1} = \text{cap}_t$, 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 $P_{11} = \text{cap}_1 - d_{21} - d_{31}$.

Period 2 Set $P_{32} = d_{32}$, $P_{12} = 0$ and $P_{22} = \text{cap}_2 - d_{32}$ provided $S_{12} > d_{12}$.

Otherwise set $P_{12} = d_{12} + d_{13} - S_{12}$,

$P_{32} = d_{32}$ and $P_{22} = \text{cap}_2 - P_{12} - P_{32}$.

Period 3 Set $P_{33} = \text{cap}_3$ 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 $P_{33} = \text{cap}_3 - P_{13} - P_{23}$.

Period 4 Set $P_{14} = \text{cap}_4$ provided $S_{24} > d_{24}$ and $S_{34} > d_{34}$.

Period 5 Set $P_{25} = \text{cap}_5$ provided $S_{15} > d_{15}$ and $S_{35} > d_{35}$.

Period 6 Set $P_{36} = \text{cap}_6$ 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 11th 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) 7th, 1st 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) 1st will be used instead of 11th 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 necessary 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 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 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

<table>
<thead>
<tr>
<th></th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>10869.0</td>
<td>15525.7</td>
<td>13082.0</td>
</tr>
<tr>
<td>50% and Iterations</td>
<td>152</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>20% util. CPU Time</td>
<td>0.94</td>
<td>16.74</td>
<td>2.14</td>
</tr>
<tr>
<td>Optimality*</td>
<td>-</td>
<td>0.70</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Current Value 18869.0
|                      | 17274.8       | 14346.5       |
| 75% and Iterations   | 152           | 667           |                    |
| 50% util. CPU Time   | 0.92          | 23.52         | 2.14               |
| Optimality           | -             | 0.62          | 0.75               |

Current Value 10898.0
|                      | 18822.2       | 15801.5       |
| 95% and Iterations   | 159           | 1013          |                    |
| 75% util. 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

<table>
<thead>
<tr>
<th></th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>73623.4</td>
<td>92583.3</td>
<td>102003.9</td>
</tr>
<tr>
<td>50% and Iterations</td>
<td>424</td>
<td>764</td>
<td></td>
</tr>
<tr>
<td>25% util. CPU Time</td>
<td>5.36</td>
<td>58.44</td>
<td>13.23</td>
</tr>
<tr>
<td>Optimality</td>
<td>-</td>
<td>0.79</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Current Value 74066.4
|                      | 92695.1       | 92639.4       |
| 75% and Iterations   | 453           | 1224          |                    |
| 50% util. CPU Time   | 6.04          | 63.74         | 13.23              |
| Optimality           | -             | 0.79          | 0.79               |

Current Value 73902.2
|                      | 97701.0       | 94046.9       |
| 95% and Iterations   | 471           | 878           |                    |
| 75% util. 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

<table>
<thead>
<tr>
<th></th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>107807.6</td>
<td>131497.9</td>
<td>115402.5</td>
</tr>
<tr>
<td>50% and Iterations</td>
<td>698</td>
<td>989</td>
<td>-</td>
</tr>
<tr>
<td>25% util. CPU Time</td>
<td>14.62</td>
<td>116.56</td>
<td>20.32</td>
</tr>
<tr>
<td>Optimality*</td>
<td>-</td>
<td>0.81</td>
<td>0.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>107889.5</td>
<td>134606.0</td>
<td>111821.5</td>
</tr>
<tr>
<td>75% and Iterations</td>
<td>600</td>
<td>1250</td>
<td>-</td>
</tr>
<tr>
<td>50% CPU Time</td>
<td>14.26</td>
<td>125.88</td>
<td>20.32</td>
</tr>
<tr>
<td>Optimality</td>
<td>-</td>
<td>0.80</td>
<td>0.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LP Solution</th>
<th>IP Solution</th>
<th>Heuristic Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Value</td>
<td>109095.4</td>
<td>134942.0</td>
<td>114784.3</td>
</tr>
<tr>
<td>95% and Iterations</td>
<td>749</td>
<td>1601</td>
<td>-</td>
</tr>
<tr>
<td>75% util. CPU Time</td>
<td>16.18</td>
<td>133.92</td>
<td>20.32</td>
</tr>
<tr>
<td>Optimality</td>
<td>-</td>
<td>0.81</td>
<td>0.93</td>
</tr>
</tbody>
</table>

* The Percentage of LP Solution
REFERENCES


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.

<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Authors &amp; Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Quantifying Message Appeals: A Decision Support System for Advertising Strategy</td>
<td>M.A.P. Davies (Jan)</td>
</tr>
<tr>
<td>2</td>
<td>The Best of British: a Peer Evaluation of Britain's Leading Companies</td>
<td>J. Saunders, M. Brown and S. Laverick (Apr)</td>
</tr>
<tr>
<td>3</td>
<td>Developing an Expert Support System for Tender Enquiry Evaluation: a Case Study</td>
<td>G. Phythian and M. King (Apr)</td>
</tr>
<tr>
<td>4</td>
<td>Validating an Expert Support System for Tender Enquiry Evaluation: a Case Study</td>
<td>G. Phythian and M. King (Apr)</td>
</tr>
<tr>
<td>5</td>
<td>Measures of Success for Management Support Systems</td>
<td>Paul Finlay, (May)</td>
</tr>
<tr>
<td>6</td>
<td>Human Jobs and Computer Interfaces. Figure-ground Reversal in Systems Development and Implementation from HCI to OSI</td>
<td>D. Buchanan (May)</td>
</tr>
<tr>
<td>7</td>
<td>Implementing Competitive Strategy: a People Centred Approach</td>
<td>P. Ahmed and M. Rafiq (May)</td>
</tr>
<tr>
<td>8</td>
<td>Research in Concert: an Evaluation of Publicity</td>
<td>Len Tiu Wright (May)</td>
</tr>
<tr>
<td>No.</td>
<td>Title</td>
<td>Authors and Dates</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>10.</td>
<td>Validity of Decision support Systems: Simplifying and Operationalising the Approach.</td>
<td>Paul Finlay and John Wilson (June)</td>
</tr>
<tr>
<td>11.</td>
<td>The UK Consumer Market</td>
<td>John Saunders and Jim Saker (July)</td>
</tr>
<tr>
<td>12.</td>
<td>Student Accommodation and Its Influence on Student Choice</td>
<td>Mark Davies, Diane Preston and John Wilson (July)</td>
</tr>
<tr>
<td>No.</td>
<td>Title</td>
<td>Author(s) &amp; Date</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Export Controls and Foreign Availability: The Case of Numerical Control</td>
<td>M. R. Hill (January)</td>
</tr>
<tr>
<td>2</td>
<td>Computer Aided Design, Manufacture and Production Management in the Soviet Union</td>
<td>M. R. Hill (January)</td>
</tr>
<tr>
<td>5</td>
<td>Entry Deterrence Through Entrant Mutual Awareness</td>
<td>Tim James (May)</td>
</tr>
<tr>
<td>6</td>
<td>In Search of a Lost Lag: The Quest for a Nest</td>
<td>P. S. H. Leeflang, G. M. Mijatovich &amp; John Saunders (May)</td>
</tr>
<tr>
<td>7</td>
<td>Product Replacement Strategies: Occurrences and Coincidences</td>
<td>John Saunders &amp; David Jobber (May)</td>
</tr>
<tr>
<td>8</td>
<td>American and Japanese HQ Views of their UK Marketing Operations</td>
<td>Len Wright, John Saunders &amp; Peter Doyle (May)</td>
</tr>
<tr>
<td>9</td>
<td>Competition in Global Markets: A Case Study of American and Japanese Competition for the British Market</td>
<td>Peter Doyle, John Saunders &amp; Veronica Wong (May)</td>
</tr>
<tr>
<td>No.</td>
<td>Title</td>
<td>Author(s) &amp; Date</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>10</td>
<td>Appraising Commercial Development Projects: Normative, Descriptive, Investigative and Prescriptive Views</td>
<td>Mohamed Kameshki &amp; John Saunders (May)</td>
</tr>
<tr>
<td>11</td>
<td>The Price, Promotion and Quality Decision within Not-For-Profit Organisations</td>
<td>Tim James &amp; John Saunders (June)</td>
</tr>
<tr>
<td>12</td>
<td>Are Asians Taking Over British Retailing?</td>
<td>Mohammed Rafiq (June)</td>
</tr>
<tr>
<td>13</td>
<td>Ethnicity and Enterprise: Muslim and Non-Muslim owned Asian Businesses in Bradford</td>
<td>Mohammed Rafiq (June)</td>
</tr>
<tr>
<td>14</td>
<td>Experiments with Expert Systems in Management Education</td>
<td>Malcolm King &amp; Lawrence McAulay (July)</td>
</tr>
<tr>
<td>15</td>
<td>Analyses of Alternative Models in Simple Bidding Situations</td>
<td>Malcolm King (July)</td>
</tr>
<tr>
<td>16</td>
<td>Property Performance Measurement</td>
<td>Paul Finlay &amp; Steven Tyler (July)</td>
</tr>
<tr>
<td>17</td>
<td>Retail Financial Services Segmentation: Time for New Tricks from Old Dogs?</td>
<td>Richard Speed &amp; Gareth Smith (July)</td>
</tr>
<tr>
<td>18</td>
<td>Recruitment Mode as a Factor Affecting Informant Response in Organizational Research</td>
<td>David Buchanan (July)</td>
</tr>
<tr>
<td>19</td>
<td>Vulnerability and Agenda: Context and Process in Project Management</td>
<td>David Buchanan (November)</td>
</tr>
<tr>
<td>20</td>
<td>Performance Measurement by Expert Assessment in the UK Retail Financial Services Industry</td>
<td>Richard Speed and Gareth Smith (December)</td>
</tr>
<tr>
<td>21</td>
<td>Evaluation Techniques Examined and Assessed: a Review of the Literature of Aids to Education</td>
<td>Alison Smith and John Piper (December)</td>
</tr>
</tbody>
</table>
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 heuristic 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.