TOPICS IN U-LINE BALANCING

By

DAVID HAMILTON SPARLING, B.SC., M.B.A.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Doctor of Philosophy

McMaster University

TOPICS IN U-LINE BALANCING

DOCTOR OF PHILOSOPHY (1997) McMASTER UNIVERSITY
(Business Administration) Hamilton, Ontario

TITLE: Topics in U-Line Balancing

AUTHOR: David Hamilton Sparling, B.SC. (Queen's University)
M.B.A. (Wilfrid Laurier University)

SUPERVISOR: Dr. G. John Miltenburg

NUMBER OF PAGES: xiv, 216

ABSTRACT

The popularity of Just-In-Time (JIT) methods in manufacturing has been rising steadily for the last two decades. U-shaped lines, or *U-lines*, are an important component of many JIT systems. While production on these lines is similar, in many respects, to that on other configurations, the U shape has significant impact on station assignments, line efficiency and production characteristics of the line. An important decision related to U-line production is the assignment of tasks to stations, the U-line balancing problem. This thesis explores aspects of U-line balancing including the nature of different U-line balancing problems, the production characteristics of U-lines and the efficiency of different solution algorithms.

The investigation into U-line balancing begins with the simplest problem, balancing a single U-line constrained only by precedence relationships and the cycle time. The efficiency of U-lines compared to straight lines and the effectiveness of different optimal solution algorithms is evaluated. An effective branch and bound algorithm is identified. It forms a basis for solving more complex problems. One of the differences between U-lines and straight lines relates to the effect of operator travel and station layout on station feasibility. Consideration of operator travel is the second topic explored. Since JIT facilities often include numerous U-lines operating in close proximity, there is an opportunity to increase operator efficiency by constructing stations which contain tasks from more than one U-line. Simultaneous balancing of more than

one line is introduced as the two U-line balancing problem and extended to N U-line balancing. If access to U-lines is restricted to the opening of each U-line, the first problem may be solved optimally for typical problems. The complexity of N U-line balancing dictates the use of heuristics solution algorithms.

The nature of JIT production frequently requires the mixing of production of several product models on a single line. One of the difficulties with mixed-model production is that different models require different production resources and task processing times vary from one model to the next. During mixed-model production station times will fluctuate, a condition termed model imbalance. Mixed-model production on U-lines is examined and solution algorithms which smooth the degree of model imbalance are offered. The particular advantages of U-lines for mixed-model production are explored.

In the final section the problem of U-line balancing is extended to include consideration of stochastic task processing times. A constrained version of this problem may be solved using dynamic programming. For general stochastic U-line balancing problems a heuristic solution is found by employing a simulated annealing algorithm.

In this thesis a number of advantages of U-lines over straight lines are illustrated. These advantages provide an incentive for the incorporation of U-lines into manufacturing processes.

ACKNOWLEDGEMENT

A Ph.D. thesis is a monumental undertaking which would be impossible without the participation of a few and the support of many. First, I would like to thank my supervisor, Professor John Miltenburg. No supervisor could be a better mix of advisor, associate and friend. His calm and insightful direction has provided both guidance and inspiration. I would also like to thank the members of my committee, Dr. George Steiner and Dr. Brian Baetz, for their time, analysis and recommendations.

Many members of my family have played a role in this educational odyssey. My parents, Hamilton and Elizabeth Sparling have always been supportive of my endeavours and may not be aware of the effect that truly great role models can play in the lives of their children. My parents-in-law, George and Margaret Kersell have, on occasions too numerous to mention, cheerfully helped with my family and business responsibilities.

This thesis is dedicated to my wife, Jane, and to our children, Beth, Laura, Julie and Cam. Jane has been my best friend and advisor through three academic programs. She has continually given her unmitigated support, without which I could never have succeeded. My children help keep the fun and laughter in my life. The love of my family gives me the happiness and balance needed to recognize what is truly important in life and to enjoy all aspects of it.

TABLE OF CONTENTS

Descriptive Note ii

Abstract iii

Acknowledgements v

Table of Contents vi

List of Figures xi

List of Tables xiii

PREFACE 1

Thesis Objectives 2

Thesis Organization 3

PART I

OPTIMAL SOLUTION ALGORITHMS FOR THE U-LINE BALANCING PROBLEM 7

Abstract 7

1. Introduction 8

2. A Model of the Simple U-Line Balancing Problem 14

3. Dynamic Programming Solution Algorithms 15

3.1 Dynamic Programming Recursion 16
3.2 Enumeration of Feasible Subsets 17
3.3 Bounds 20
3.4 Computational Complexity 22
4. Branch and Bound Solution Algorithms 23
4.1. Breadth-first Solution Algorithm 23
4.2. Depth-first Solution Algorithm 29
5. Empirical Results 31
6. Summary and Conclusions 40
References 43
Appendix 1 A Reaching Dynamic Programming Example 46
Appendix 2 A Bound on the Number of Feasible Subsets 49
Appendix 3 Procedure for Enumerating Successor Nodes 53
Appendix 4 Procedure For Randomly Generating Line Balancing Problems 56

PART II

EXTENDING THE U-LINE BALANCING PROBLEM:
TRAVEL DISTANCES AND BALANCING TWO U-LINES 58
Abstract 58
1. Introduction 59
2. The Simple U-line Balancing Problem with Travel 66
2.1 Definitions and Notation 66
2.2 Calculating Travel Distance and Travel Time 69
2.3 A Model for the SULB-T Problem 70
2.4 A Branch and Bound Algorithm 71
3. The Two U-line Balancing Problem 75
3.1 Definitions and Notation 75
3.2 A Model for the 2ULB-T Problem 78

3.3 An Optimal Solution Algorithm For The 2ULB-T Problem 79
3.4 Computational Complexity 86
4. Computational Results 87
5. Discussion and Conclusions 98
5.1 Extensions 100
References 104
Appendix An Illustrative Example of the 2ULB-T Problem 106

PART III

BALANCING JUST-IN-TIME PRODUCTION UNITS:
THE N U-LINE BALANCING PROBLEM 111
Abstract 111
1. Introduction 112
2. Modelling the NULB-T Problem 118
2.1 Assumptions 118
2.2 Notation 120
2.3 NULB-T Model 121
3. A Solution Algorithm for the NULB-T Problem 122
3.1 Selecting Multi-line Subsets 123
3.2 The Multi-line Station Assignment Problem - General Case 126
3.3 The Multi-line Station Assignment Problem
- Fixed U-line Locations 138
3.4 A Comparison of Complexity of Line Balancing Problems 141
4. A Computational Study of the NULB-T Problem 141
4.1 Study Parameters and Procedure 141
4.2 Study Results 145
5. Discussion 148
5.1 Extensions to NULB-T Research 149

References 152
Appendix Bounds on the Sum of Task Times in a Multi-Line Subset 153

PART IV

MIXED-MODEL LINE BALANCING:
U-LINES AND SMOOTHING STATION ASSIGNMENTS 155
Abstract 155
1. Introduction 156
2. Review of Mixed Model Line Balancing for a Straight Line 158
3. The Mixed-Model U-Line Balancing (MMULB) Problem 163
    3.1 Notation 163
    3.2. A Model for the MMULB Problem 165
    3.3. A Solution Algorithm for the MMULB Problem 166
    3.4 A Smoothing Algorithm for the MMULB Problem 167
4. Discussion and Summary 175
References 178
Appendix The Mixed-Model Multiple U-Line Balancing (N-MMULB) Problem . 180

PART V

STOCHASTIC U-LINE BALANCING 184
Abstract 184
1. Introduction 185
2. A Dynamic Programming Approach For Solving the Stochastic U-Line
    Balancing Problem 188
    2.1 Notation 190
    2.2 A Model For The Stochastic U-Line Balancing Problem 191
    2.3 A Dynamic Programming Algorithm For Stochastic U-Line

Balancing 191
2.4 Consideration Of Station Travel 197
3. A Simulated Annealing Approach To Stochastic U-line Balancing 199
3.1 Notation 199
3.2 Objectives for Simulated Annealing 199
3.3 A Simulated Annealing Algorithm for the STULB Problem 200
3.4 The Impact of Exceeding Cycle Time on a U-Line 203
4. Unbalancing In Stochastic Assembly Line Balancing 204
5. Discussion 206
References 207

Concluding Remarks 210

Extensions To U-Line Balancing Research 213

LIST OF FIGURES

PART I

Figure 1 Straight lines and U-Lines 10
Figure 2 Precedence Graph for the Lines in Figure 1 12
Figure 3 Precedence Relations for Example 1 27
Figure 4 Illustrative Example of the Branch and Bound Algorithm 28
Figure 5 Bound on the Number of Feasible Subsets 50

PART II

Figure 1 Traditional Line and U-Lines 60
Figure 2 An Instance of a Multiple U-Line Balancing Problem 61
Figure 3 U-Line with Travel Distance: Notation and Terminology 67
Figure 4 Two U-Lines with Multi-Line Stations 76
Figure 5 The Effect of Order Strength on Beta Selection 92
Figure 6 The Effect of Altering the Travel Time/Task Time Ratio 93
Figure 7 Optimal Balance for Illustrative Example 110

PART III

Figure 1 Two U-Lines with Multi-Line Stations 115
Figure 2 An Example of a Just-In-Time Production Unit 117
Figure 3 Precedence Graphs for Example 1 133
Figure 4 Example of a General NULB-T Balance with Divided Subsets 137
Figure 5 Example of a NULB-T Balance with Fixed Locations 142

PART IV

Figure 1 Precedence Graph and Straight Line Balance for Example in Section 2 . 160
Figure 2 Mixed-Model Production on a U-Line 164

Figure 3 Initial Balance for Example in Section 3 171

PART V

Figure 1 A Typical U-Line Balance 189
Figure 2 Precedence Relations for Example 1 195

LIST OF TABLES

PART I

Table 1 Results from Part 1 of the Empirical Study 33
Table 2 An Illustrative Example of the Reaching DP Algorithm 47
Table 3 Enumerating Nodes for the Example in Figure 3 55
Table 4 Random Problem Generation Parameters 57
Table 5 Problem Set Characteristics 57

PART II

Table 1 Problem Set Characteristics and Study Parameters 89
Table 2 Three Methods of Implicitly Including Travel in U-Line Balancing 90
Table 3 Results from 50 U-Line Balancing Problems, each with Ten Tasks 91
Table 4 Problems in the Second Study 96
Table 5 Data for the Illustrative Example 107

PART III

Table 1 Results from Steps 1 and 2 of the NULB-T Algorithm 134
Table 2 Multi-Line Station Assignments After Heuristic 1 136
Table 3 Computational Complexity of Different Line Balancing Problems 143
Table 4 Selected Results for 100 NULB-T Problems Solved Using Criterion 1 . . 146

PART IV

Table 1 Task Data for Example in Section 2 159
Table 2 Total Time Required by each Model 162
Table 3 Calculations for MMULB Example in Section 3 172
Table 4 Task Processing Time Variance and Correlation Coefficients for
Example in Section 3 173

PART V

Table 1 An Illustrative Example of the Stochastic DP Algorithm 196
Table 2 Dominance in Duplicate Subsets of Size n=3 in the Example 198

PREFACE

To succeed in the current competitive environment manufacturing organizations must operate with smaller inventories, shorter lead times and have the ability to respond rapidly to changes in demand. These companies must meet customer expectations for continuous improvement in the value of their products by increasing quality and through vigorous cost reduction efforts. For many organizations Just-In-Time (JIT) production is seen as the means of meeting these challenges. The move to JIT involves major changes in manufacturing operations: production switches from a push to a pull system; batch sizes are reduced, with many facilities using mixed-model production; and layouts are altered, often incorporating U-shaped production lines. It is the last factor, the use of *U-lines*, which is the focus of this thesis.

U-lines are one element of *Shojinka*, the continuous effort to reduce waste in manufacturing. Combined with the cross-training of operators and the continual review and revision of work practices, U-lines contribute to improved manufacturing efficiency and effectiveness. The shape of a U-line provides two benefits: better visibility and greater line balancing flexibility. If an operator in one station experiences difficulties other operators on the U-line can easily be alerted and provide assistance. A central question to manufacturing on any type of line is how to assign operators to stations, the *Line Balancing Problem*. On a U-line a station may be organized so that during a single

cycle an operator can complete tasks on the front of the line, cross to the back of the line, complete tasks there and return to the beginning of the station (see stations 1 and 2 in Figure 1 on page 10). A U-line provides greater flexibility in station construction than a straight line and the number of operators needed will be less than or equal to the number required on an equivalent straight line.

This thesis considers several topics related to the problem of assigning tasks to stations on U-lines, the *U-Line Balancing (ULB) Problem.* Although the production process on a U-line is similar to that on a straight line, several characteristics of U-lines and their use in JIT systems make balancing U-lines a significantly different problem from straight line balancing.

Thesis Objectives

Little research exists on the ULB problem. The primary goal of this study is to expand our understanding of U-line balancing and to explore how it differs from straight line balancing. Each of the five papers forming this thesis considers different aspects of U-line balancing. Two questions remain central to the investigation.

1) How does station construction and operation differ between U-lines and straight lines?

2) What effect do the differences between U-lines and straight lines have on manufacturing operations?

A secondary thesis objective is to present at least one algorithm for solving each of the problems examined. Where possible the algorithms seek an optimal solution.

Thesis Organization

The analysis begins with the most basic ULB problem. Additional constraints and capabilities are incorporated into successive versions, gradually building toward more complex and realistic problem variations. The body of the thesis is organized into five sections, each presented as a separate and complete paper prepared for publication. The five papers are briefly described.

Part I. Optimal Solution Algorithms For The U-Line Balancing Problem

The first problem examined is one of balancing a single U-line with stations constrained by cycle time and precedence relationships only. This *Simple U-Line Balancing* problem is equivalent to the *Simple Assembly Line Balancing* problem on a straight line. Different optimal solution algorithms for the problem are presented and their effectiveness is evaluated in a computational study.

Part II. Extending The U-Line Balancing Problem:

Travel Distances and Balancing Two U-Lines

Station travel on a straight line is a function of task length. Time for this travel may be implicitly included in the task processing times and has no further impact on station feasibility. In contrast, travel in a U-line station varies depending on task locations and line layout. Station feasibility is affected by both travel time and travel paths. To ensure station feasibility, the SULB problem is extended to include consideration of station travel time and task locations. An

optimal branch and bound algorithm for this *Simple U-Line Balancing with Travel* problem is offered.

In some JIT facilities a number of U-lines operate in close proximity. The opportunity exists to balance two or more U-lines simultaneously and reduce the total number of operators required to operate the facility. The topic of multiple U-line balancing is introduced through the *Two U-Line Balancing with Travel* problem. An optimal branch and bound algorithm is offered and the effectiveness of balancing two U-lines simultaneously is examined in a computational study.

Part III. Balancing Just-In-Time Production Units: The N U-Line Balancing Problem

Multiple line balancing is extended to balancing more than two U-lines simultaneously. The complexity of this *N U-Line Balancing with Travel* problem is such that finding an optimal solution is not feasible for most problems. Heuristic algorithms are offered for both the general case, where U-line locations are not fixed, and for the fixed location problem. The effectiveness of multiple line balancing is evaluated through a computational study.

Part IV. Mixed-Model Line Balancing: U-Lines and Smoothing Station Assignments

JIT manufacturing frequently requires the production of several product models on a single line. *Mixed-model* production is scheduled so that models are interspersed throughout the production sequence in proportion to their demand. Since different models require different task combinations and task times, total station times may vary significantly as the models flow along the line. A problem

for mixed-model line balancing is to minimize the impact of these variations, a process termed *smoothing the line balance*. Algorithms are presented for mixed-model line balancing with smoothing on single and multiple U-lines.

Part V. Stochastic U-Line Balancing

The line balancing problems in the first four parts of the thesis all assume deterministic task processing times. In many production facilities this is not a valid assumption. The problem of balancing U-lines with stochastic task processing times is analysed. For one well defined formulation of the problem, an optimal solution may be found using dynamic programming. For more general stochastic U-line balancing problems a statistical search heuristic, simulated annealing, is employed.

In this thesis, it will be shown that U-lines offer several advantages over straight production lines. While a higher level of manufacturing capability is required to successfully utilize U-lines, the potential benefits provide a strong incentive for integrating U-lines into manufacturing facilities. However, the applicability of the research and algorithms presented is not limited exclusively to U-lines. Any production configuration where different lines, or line segments, are located close together provides an opportunity to construct stations which include previously unrelated tasks. Where this occurs, the methods proposed for U-line balancing may be employed to capture some of the advantages offered by U-lines and thereby improve manufacturing performance. Directions for this and other future research are discussed.

All algorithms, analysis and computational studies are the work of the author, with guidance and suggestions by Dr. G. J. Miltenburg in areas of model and algorithm development, computations, analysis and organization.

OPTIMAL SOLUTION ALGORITHMS FOR THE U-LINE BALANCING PROBLEM

Abstract

The simple U-line balancing (SULB) problem assigns a partially ordered set of tasks to stations on a U-shaped production line. U-line facilities consist of many individual U-lines, each typically having between 5 and 20 tasks. The traditional simple assembly line balancing (SALB) problem, where stations are arranged on a straight production line, is just a simple case of the SULB problem. Three optimal solution algorithms -- a dynamic programming algorithm, a breadth-first branch and bound algorithm and a depth-first branch and bound algorithm -- are presented in this paper. The complexity of the dynamic programming algorithm is determined. The optimal solution algorithms are capable of solving realistic problems. Insights into the SULB problem and algorithm performance are developed through an empirical study.

This paper has undergone two revisions for IIE Transactions and has been resubmitted for review.

1. Introduction

Manufacturers have traditionally used *straight* assembly lines (or fabrication lines) to produce high-volume products. These lines consist of stations where production tasks are completed on the product as it moves from station to station down the line. One of the problems associated with designing and operating assembly lines is determining how to assign tasks to stations. The simple assembly line balancing (SALB) problem assumes a single product, a precedence graph, deterministic task processing times and a specified cycle time, and assigns tasks in such a way that the number of stations is minimized.[1] This problem, along with numerous optimal and heuristic solution algorithms, is the subject of a considerable literature. Some of the more recent papers include Baybars [1986], Ghosh and Gagnon [1989], Hackman, Magazine and Wee [1989], Johnson [1981, 1988a, 1988b], Talbot and Patterson [1984], Talbot, Patterson and Gehrlein [1986], and Hoffmann [1992].

Recently manufacturers have begun using *U-shaped lines* rather than straight lines, most often as a consequence of the implementation of just-in-time (JIT) principles into their operations. In his book, *The Toyota Production System,* Monden [1993] describes the many elements that comprise just-in-time, including the use of U-lines. See also Hall [ch. 6, 1983] and the Japan Management Association [pp. 122-125, 1987]. Miltenburg

---

[1] This is actually the SALB-I problem. There is an SALB-II problem that seeks to minimize the cycle time for a given number of stations.

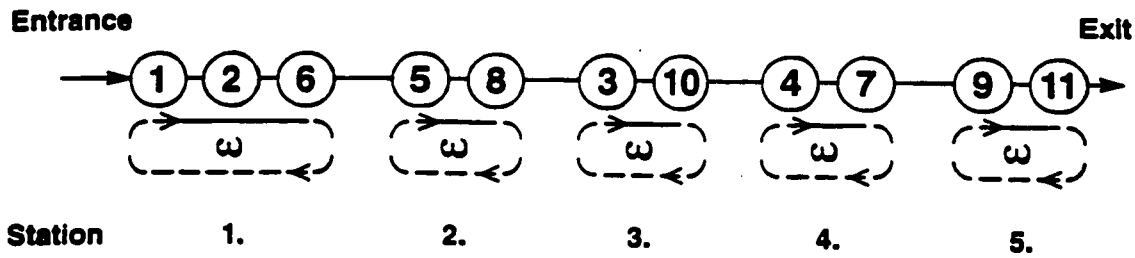
and Wijngaard [1994] presented a definition, model and solution algorithms for the problem of assigning tasks to stations on this new line. They call the resulting problem the simple U-line balancing (SULB) problem. As the goal is to minimize the number of stations for a given cycle time, the SULB problem corresponds to the SALB problem.

Figure 1 shows a traditional straight-line and a U-line. Notice that on a straight-line the tasks that comprise a station are located one after the other. In this example, the operator at the first station performs task 1, then task 2 and finally task 6 at the end of the station. He/she then returns to the beginning of the station and starts again. In the U-line, stations are comprised of tasks that may be located on opposite sides of the line (such as stations 1 and 2 in Figure 1), as well as tasks located one after the other (stations 3 and 4). It will be shown that the straight-line is a special case of the U-line. Notice also that operators are located inside the U-line, in close proximity to each other. This improves communication and increases visibility. As operators are multi-skilled, it is relatively easy for them to work together to identify and solve problems.

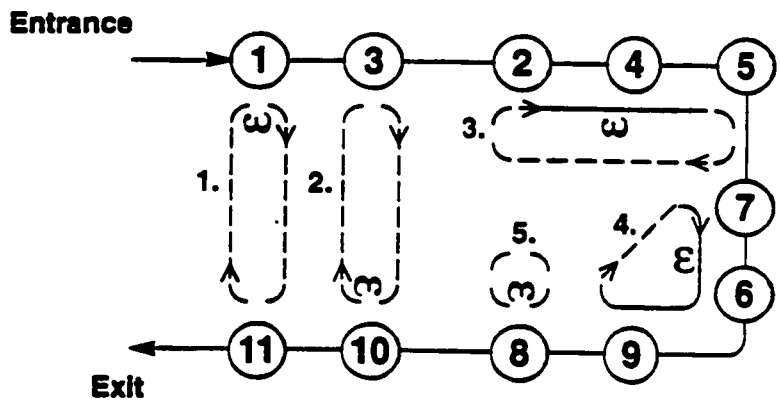
One feature of JIT is that the production rate changes when the demand changes (to avoid the buildup of inventory). The effect of this on the U-line is the following. The production rate of any line is the cycle time, specifying, as it does, the time between the completion of consecutive products. The cycle time is also the upper bound on the time a station has available for completing its tasks. When the demand changes, the cycle time changes, which may require a change in the assignment of tasks to stations (that is, a re-balancing of the line). Because the U-line offers more options for assigning tasks to stations, it is easier to modify an existing balance for a U-line than it is for a straight-line.

Figure 1

Straight line and U-Line Production Processes



Straight-Line



U-Line

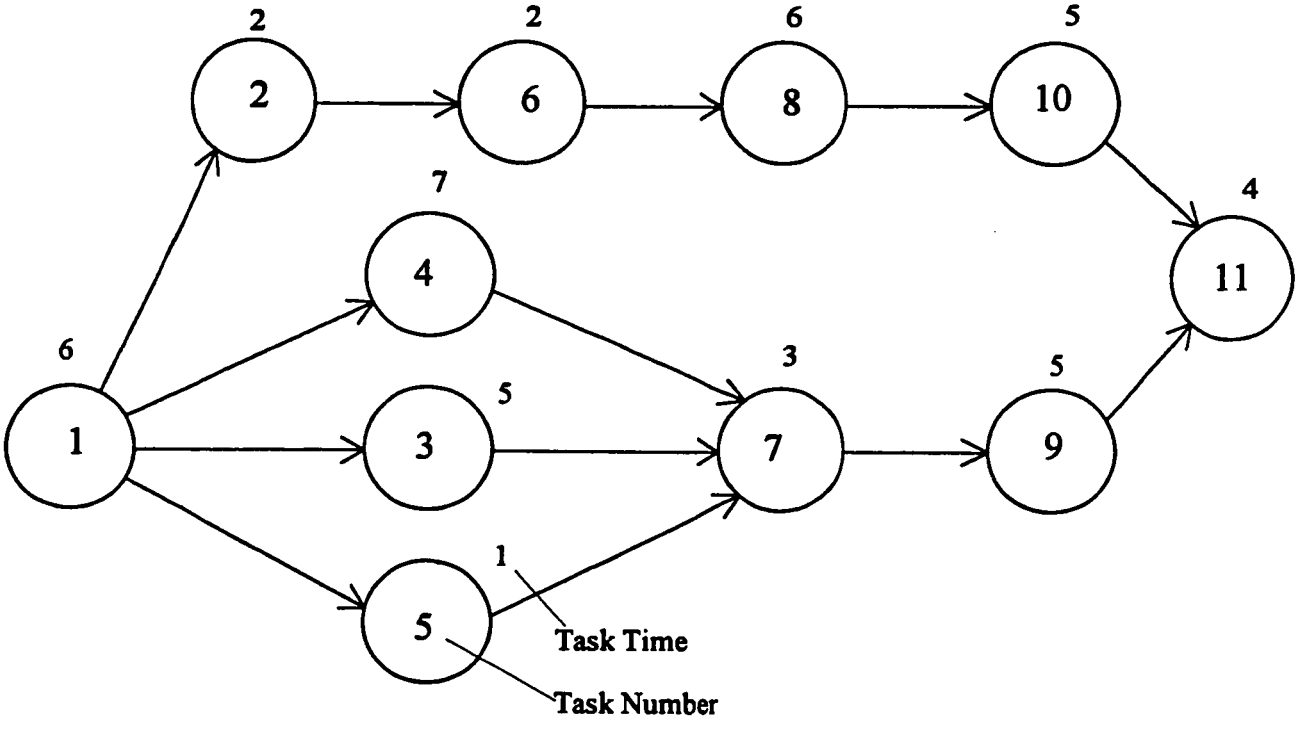
Consequently, the minimum number of stations on a U-line will be less than or equal to the minimum number of stations on a straight-line.

The precedence graph for the lines in Figure 1 is shown in Figure 2. (This is Jackson's 11-task problem [1956].) Notice that stations on the straight-line consist of tasks whose predecessor tasks have been assigned to the same station or to earlier stations. For example, task 8 is the last task in station 2. Its predecessors, tasks 1, 2 and 6, are assigned to an earlier station--station 1. Stations on a straight-line may be formed by moving through the precedence graph in a forward (or backward) direction and assigning tasks to stations whose predecessors (successors) have already been assigned. Stations on the U-line are different. The tasks that comprise a station on a U-line are those whose predecessors or successors have already been assigned to the same station or to earlier stations. For example, task 10 is the last task in station 2 on the U-line in Figure 1. While three of its predecessors, tasks 2, 6 and 8, have not yet been assigned, its successor, task 11, is assigned to an earlier station--station 1. Stations on a U-line may be formed by moving simultaneously through the precedence graph in a forward direction and a backward direction, assigning tasks whose predecessors or successors have already been assigned. (See Miltenburg and Wijngaard [1994].)

The SALB problem is well known to be an NP-hard problem. The SALB problem is a special case of the SULB problem where tasks can only be assigned working through the precedence relationships in one direction. Thus the SULB problem is also NP-hard. The limits on finding an optimal solution for instances of both problems depend on several factors including the number of tasks, the width and density of the precedence graph,

Figure 2

Precedence Graph for the Lines in Figure 1



the distribution of task processing times, and the cycle time. It will be shown that computation time and storage space requirements for the SULB problem are significantly higher than for the SALB problem, and that optimal solution algorithms are unable to solve certain difficult instances, such as those with more than 30 tasks, within a reasonable time. This limitation is not prohibitive, however, because most U-lines are small, typically consisting of between 5 and 20 tasks.

Optimal solution algorithms for the SALB problem and the SULB problem fall into two categories, dynamic programming (DP) and branch and bound (B&B) algorithms. Dynamic programming algorithms use either a recursive fixing procedure (Schrage and Baker [1978]) or a reaching procedure (Lawler [1978], Denardo and Fox [1984]) to enumerate feasible subsets, and apply the DP recursion to evaluate the cost of the feasible subsets. Miltenburg and Wijngaard [1994] presented a recursive fixing DP algorithm for the SULB problem and used it to solve instances with up to ten tasks. (They also presented two heuristics--a modified Hoffmann enumeration heuristic and a ranked positional weight heuristic--for solving larger instances.) Branch and bound algorithms are of two types, breadth-first and depth-first. Various solution algorithms for the SALB problem were reviewed by Johnson [1981], Talbot and Patterson [1984], Queyranne [1985] and Hackman, Magazine and Wee [1989]. Computational comparisons in these papers conclude that B&B algorithms are more effective than DP algorithms, and that the breadth-first algorithm of Hackman, Magazine and Wee [1989] and the depth-first algorithm of Talbot and Patterson [1984] are among the particularly effective algorithms. More recently, two depth-first algorithms, Fable by Johnson [1988] and Eureka by

Hoffmann [1992], have been shown to be effective at solving SALB problems with hundreds of tasks.

This paper extends the previous work on the SULB problem in three ways. First, a reaching dynamic programming algorithm is presented. This algorithm is shown to be more effective than the recursive fixing dynamic programming algorithm and permits optimal solutions to be obtained for problems with up to 30 tasks. Second, the complexity of the dynamic programming algorithm is determined and bounds are placed on the number of feasible subsets that are enumerated. Third, two B&B algorithms are developed, a breadth-first algorithm and a depth-first algorithm. The breadth-first algorithm is shown to outperform the dynamic programming and depth-first algorithms.

The remainder of the paper is organized as follows. A model of the SULB problem is reviewed in section 2. Dynamic programming and B&B algorithms are presented in sections 3 and 4. Results of an empirical study are described in section 5. A summary and discussion of areas where more research is needed are given in section 6.

2. A Model of the Simple U-Line Balancing Problem

The following model was presented in Miltenburg and Wijngaard [1994].

N = The number of tasks to be completed on the U-line.

$S = \{x \mid x=1,\ldots,N\}$. The set of tasks.

t_x = The (deterministic) processing time for task x.

$P = \{(x,y) \mid$ task y is an immediate successor of task x$\}$. The set of precedence constraints.

C = The cycle time.

ST_m = {x | task x is performed at station m}. The tasks assigned to station m.

M = The number of stations.

The SULB problem is:

Minimize M

Subject to:

$$\bigcup_{m=1}^{M} ST_m = S$$

$$ST_u \bigcap_{u \neq v} ST_v = 0$$

$$\sum_{x \in ST_m} t_x \leq C \qquad m = 1,\ldots,M$$

For every task $y \in S$

Either: If $(x,y) \in P$, $x \in ST_i$, $y \in ST_j$, then $i \leq j$, for all x;

Or: If $(y,z) \in P$, $y \in ST_j$, $z \in ST_k$, then $k \leq j$, for all z.

The objective is to minimize the number of stations, M. This is equivalent to minimizing the total idle time, $MC - \sum_{x \in S} t_x$, since C and $\sum_{x \in S} t_x$, are constants. The first constraint ensures that all tasks are assigned to a station. Tasks are assigned to only one station as a consequence of the second constraint. The sum of the task times at a station cannot exceed the cycle time because of the next constraint. The fourth constraint ensures that tasks will be assigned to stations only when all their predecessors or successors have already been assigned to the same station or to a previously constructed station.

3. Dynamic Programming Solution Algorithms

Dynamic programming algorithms find the optimal sequence in which the N tasks are assigned to the M stations. The algorithms consist of two procedures, the recursion

and the generation of feasible subsets. The dynamic programming recursion is discussed in section 3.1. Two approaches for generating feasible subsets, recursive fixing and reaching, are outlined in section 3.2. Bounds on the number of feasible subsets are presented in section 3.3. The complexity of the dynamic programming algorithm is determined in section 3.4.

3.1 Dynamic Programming Recursion

Define the following notation.

$S_i \subset S$ — S_i is a feasible subset of S if and only if for every task, $x \in S_i$, all predecessors or all successors of x are in S_i.

$k =$ — A task which when removed from S_i leaves the subset S_i-{k} feasible.

$I_m =$ — Idle time at station m.

$F[S_i] =$ — The minimum cost of feasible subset S_i. Cost is defined as the number of "full" stations (those where there is no available task x with $t_x \leq I_m$) required to complete the tasks in S_i plus the time in last station. The minimum cost is achieved when tasks are arranged in an optimal sequence and assigned to stations in that sequence.

$g(k) =$ — The additional cost incurred when task k is added to the end of the optimal sequence of the tasks in S_i-{k}.

The following dynamic programming recursion is applied to every feasible subset, S_i.

$$F[S_i] = \min_{k \in Q(S_i)} \{ F[S_i], F[S_i\text{-}\{k\}] + g(k) \}, \quad Q(S_i) = \{k \in S_i, S_i\text{-}\{k\} \text{ is feasible}\}$$

$$F[\varnothing] = 0$$

The last feasible subset to be enumerated is S, the complete set of tasks. When this is done, the optimal assignment of tasks to stations is constructed by backtracking through $F[S_i]$ to determine the optimal sequence of tasks (x(N), x(N-1), ..., x(1)), where task x(j) is the j'th task in the sequence. Tasks are then assigned to stations in reverse order, (x(1), x(2), ..., x(N)), until each station is filled.

3.2 Enumeration of Feasible Subsets

Recursive Fixing Approach

Miltenburg and Wijngaard [1994] used the recursive fixing approach (see Schrage and Baker [1978]) in their dynamic programming algorithm. They define forward and backward feasible subsets, S_i^f and S_j^b. A subset $S_i^f \subset S$ is forward feasible if and only if for every task, $x \in S_i^f$, all predecessors of x are also in S_i^f. A subset $S_j^b \subset S$ is backward feasible if and only if for every task, $y \in S_j^b$, all successors of y are also in S_j^b. Their algorithm may be described as follows.

S_j^b Loop:

- Select the next backward feasible subset, S_j^b (by moving through P in a backward direction).

S_i^f Loop:

- Select the next forward feasible subset, S_i^f, (by moving through P in a forward direction) where no task $x \in S_i^f$ is already in S_j^b and where the subset $S_i = S_j^b \cup S_i^f$ was not already considered during this loop.
- Perform the DP recursion:
  $F[S_i] = \min_{k \in Q(S_i)} \{F[S_i], F[S_i-\{k\}] + g(k)\}$, $Q(S_i) = \{k \in S_i, S_i-\{k\} \text{ feasible}\}$

End of S_i^f Loop.
End of S_j^b Loop.

For the SALB problem, Schrage and Baker [1978] provide a compact scheme for assigning unique labels to subsets. It takes advantage of two features of the SALB problem; 1) the task numbering scheme whereby if task i precedes task j then $i < j$, and 2) the problem is solved by working through the precedence relations in one direction (either forward or backward). As neither feature is found in the SULB problem, the labelling scheme cannot be extended to this problem. (It would, for example, assign the same label to different subsets.) Consequently, in this paper a binary labelling scheme is used for the SULB problem.

Let $\#(S^f)$ and $\#(S^b)$ denote the number of forward and backward feasible subsets, each including the null set. If S_i is a forward feasible subset then its complement $\{S\text{-}S_i\}$ must be a backward feasible subset. Similarly, the complement of each backward feasible subset is a forward feasible subset. Therefore, $\#(S^f) = \#(S^b)$, and a loose upper bound on the total number of feasible subsets for the SULB problem is $[\#(S^f)]^2$.

The drawback of the recursive fixing approach for the SULB problem is that subsets may be considered more than once, and, because of the way the subsets are enumerated, all the data required to evaluate each $S_i - \{k\}$ will not be available until the last time each subset S_i is enumerated.

Reaching Approach

The reaching approach (Lawler [1979]) is an improvement over the recursive fixing approach because each feasible subset is considered at one stage only and all data required

to determine $F[S_i]$ is available at that time. The reaching approach generates subsets in increasing order of subset size. Define $S_i^{(n)}$ to be the i'th feasible subset of size n tasks, and $\#(S_i^{(n)})$ to be the number of such subsets. Also define an assignable task, u(j), j=1,2,..., for subset $S_i^{(n)}$ to be a task that is not an element of $S_i^{(n)}$, but whose predecessors or successors are all elements of $S_i^{(n)}$. Finally let $S_{i,j}^{(n+1)}$ be a feasible subset of size n+1 formed by adding one assignable task, u(j), to the end of $S_i^{(n)}$; $S_{i,j}^{(n+1)} = S_i^{(n)} + \{u(j)\}$. Then the reaching DP algorithm is implemented as follows.

Set n=0, $F[S_i^{(0)}]=0$.

n Loop: *Generate all feasible subsets of size n = 1 to N.*

- Set n=n+1, i=0.

  Set List(a)=$\varnothing$, List(b)=$\varnothing$.

i Loop: *For each subset i, of size n-1, generate corresponding subsets of size n.*

- Set i=i+1, j = 0.
- When $i > \#(S_i^{(n-1)})$, exit i Loop.

j Loop: *For a subset $S_i^{(n-1)}$, generate all feasible subsets $S_{i,j}^{(n)} = S_i^{(n-1)} + \{u(j)\}$.*

- j = j+1.
- Find the next task u(j) which is assignable and add u(j) to the end of feasible subset $S_i^{(n-1)}$; $S_{i,j}^{(n)} = S_i^{(n-1)} + \{u(j)\}$. Calculate the cost $F[S_{i,j}^{(n)}] = F[S_i^{(n-1)}] + g(u(j))$.
- For each j save $S_{i,j}^{(n)}$, u(j) and $F[S_{i,j}^{(n)}]$ in List(b), the current subset data array.
- The j loop ends when there are no more tasks assignable to subset $S_i^{(n-1)}$.

End of j loop

Merge List(a) and List(b)

- When List(b) is completed, merge it with List(a), the master

subset data array. During merging, occurrences of the same subset in both lists are identified and only the one with the minimum cost is retained in List(a). Ties are broken by retaining the subset from List(a) only.

- Set List(b)=$\varnothing$.

End of i Loop.

- Copy the results in List(a) to List(n), the list of subsets of size n. Set List(a) = $\varnothing$

End of n Loop.

When the set of all tasks, S, is enumerated, the optimal sequence of tasks is constructed by backtracking. An illustrative example is solved in Appendix 1.

3.3 Bounds

To estimate the computation time and storage space requirements for the SULB problem, it is necessary to determine the number of feasible subsets. While this is a #P-complete problem (Provan and Ball [1983]), it is possible to determine an upper bound on the number of feasible subsets.

Bounds for the SALB problem

Schrage and Baker [1978] developed a labelling scheme that can be used in the SALB problem to assign a label, L_x, to each task x. The sum of the labels, $\sum_{x \in S} L_x$, is an upper bound on the actual number of forward feasible subsets, $\#(S^f)$. Subsequent work by Kao and Queyranne [1982] showed that the difference between this bound and the actual

number of subsets, $\sum_{x \in S} L_x$-$\#(S^f)$, grows exponentially as the order strength of the problem decreases. Order strength is defined as the number of precedence relations, R, in the precedence graph divided by the maximum number of possible precedence relations, N(N-1)/2; order strength = R/[N(N-1)/2].

Steiner [1990] considered the effect of the width of the precedence graph, w, on the number of (forward) feasible subsets. He defined width as follows. For any $u,v \in S$, if u must precede v in every feasible sequence (that is, task u must be completed before task v), then $u <_p v$ and, u and v are comparable. A subset $C \subset S$ is a chain if for every $u,v \in C$, u and v are comparable. A subset $AC \subset S$ is an antichain if no two elements of AC are comparable. The maximum size of an antichain in P is the width, w, of P. Steiner's bound on $\#(S^f)$, the number of forward feasible subsets, is, $\#(S^f) \leq [(N+w)/w]^w$.

Bounds for the SULB problem

A very loose upper bound on the number of feasible subsets for the SULB problem, #(S), is found by squaring one of the bounds on the number of forward feasible subsets--the Schrage and Baker bound or the Steiner bound. If $\#(S^f)$ was determined for the less complex SALB problem then a tighter lower bound is provided by $\#(S) \leq \#(S^f)^2$. Using this relationship, both previous lower bounds may be tightened further as a consequence of the following lemma, the proof of which is given in Appendix 2.

Lemma: An upper bound on the number of feasible subsets, #(S), for the SULB problem is $\#(S) \leq [\#(S^f)^2+\#(S^f)]/2$.

Since the Schrage and Baker bound and the Steiner bound are upper bounds on

$\#(S^f)$, they may be tightened in the same manner. These bounds are more useful since the SALB problem need not be solved prior to calculating the bound. In summary, the upper bounds on the number of feasible subsets for the SULB problem are:

$\#(S) \le [(\sum_{x \in S} L_x)^2 + \sum_{x \in S} L_x]/2$,

$\#(S) \le [((N+w)/w)^{2w} + ((N+w)/w)^w]/2$, or

$\#(S) \le [\#(S^f)^2 + \#(S^f)]/2$, which cannot be used unless $\#(S^f)$ is found first.

3.4 Computational Complexity

The complexity of the dynamic programming algorithms is a function of the number of feasible subsets, $\#(S)$, that are generated. Unfortunately, as determining $\#(S)$ is itself an #P-complete problem, the best that can be done is to express the complexity as a function of an upper bound on $\#(S)$; say $\#(S) \le [\#(S^f)^2 + \#(S^f)]/2$. Thus $\#(S)$ may be considered to be $O(\#(S^f)^2)$.

Under the reaching approach a single task, u, is added to each feasible subset, provided all of its predecessors or successors are already in the subset. As there are N tasks, each of which requires $O(N^2)$ computations to check for predecessors and successors, the complexity of dynamic programming under the reaching approach is $O(N^2[\#(S^f)^2])$. The complexity is the same when the recursive fixing approach is used because, rather than add a task, u, the recursive fixing approach removes a single task, k.

It is interesting to note that the complexity of dynamic programming for the SALB problem is $O(N^2[\#(S^f)])$ (Kao and Queyranne [1982]). Clearly the SULB problem is a very difficult problem.

4. Branch and Bound Solution Algorithms

A number of empirical studies of the SALB problem concluded that B&B algorithms out-performed dynamic programming algorithms. See, for example, Johnson [1981] and Talbot and Patterson [1984]. Recent efforts have shown that both breadth-first (Hackman, Magazine and Wee [1989]) and depth-first (Talbot and Patterson [1984], Hoffmann [1988] and Johnson [1988]) algorithms are effective for the SALB problem. Because of the nature of the SULB problem, where tasks may be assigned from both ends of the precedence graph, not all SALB algorithms translate effectively to the SULB problem. A breadth-first algorithm and a depth-first algorithm are now described for the SULB problem.

4.1. Breadth-first Solution Algorithm

The breadth-first B&B approach works well for the SULB problem. A particularly effective algorithm, which builds on work of Van Assche and Herroelen [1978], Johnson [1988] and Hackman, Magazine and Wee [1989] for the SALB problem, consists of the following steps.

Step 1: Initial Bounds on the Number of Stations

1.1 An Upper Bound

Solve the SULB problem using a heuristic, H, to obtain an upper bound, U_H, on the number of stations. Call this assignment of tasks to stations the current best solution, $U_{current} = U_H$. A single pass, maximum task time heuristic was used in this

research because it is simple, fast and effective (Hackman, Magazine and Wee [1989]). Tasks from a set of available tasks are assigned to stations in decreasing order of task time and the set of available tasks is updated each time a task is assigned.

1.2 A Lower Bound

Determine the theoretical lower bound on the number of stations, $L^* = \lceil \sum_{x \in S} t_x / C \rceil$.[2] If $U_{current} = L^*$ then stop.

***Step 2:** Branch and Bound*

2.1 Enumerate Successor Nodes (Stations) to the Branching Node

- Except for the first node, which is the starting node, each node represents a station. In this step all nodes that follow the current node are enumerated using the node enumeration procedure described Van Assche and Herroelen [1978] (see Appendix 3).
- As successor nodes to node i are created, any nodes which are subsets of other successor nodes of i are fathomed (Jackson [1956]).
- When each node is created, a label representing the set of assigned tasks in the path to and including the node, is computed. Once all successor nodes to a node i have been enumerated their labels are compared to labels of previously created nodes. Where nodes have the same label, the node with the fewest number of stations is retained, and the other nodes, with their successors, are fathomed. In

---

[2] $\lceil z \rceil$ is the smallest integer $\geq z$ and $\lfloor z \rfloor$ is the integer portion of z.

the event of a tie, the first node that was generated is retained. This corresponds to Johnson's [1988] labelling dominance rule, with the difference that a binary labelling scheme is used instead of Schrage and Baker's [1978] labelling scheme.

• If a node completes a solution to the problem which improves on the current best solution update $U_{current}$.

2.2 Determine the Lower Bound for each Node

• For each node, i, calculate the accumulated idle time, I_i, in the path to the node, as well as a lower bound, LB_i, on the number of stations. The following four lower bounds are determined. (LB2 is new, LB3 and LB4 were used by Johnson [1988].)

Lower Bound 1:

If D_i is the depth of node i (that is, the number of nodes or stations in the path to and including node i) then the theoretical lower bound is

$$LB1_i = D_i + \lceil \sum_{x \in A_i} t_x/C \rceil,$$

where A_i is the set of tasks not yet assigned to stations.

Lower Bound 2:

If t_{min} is the average of the two minimum unassigned task times and $\#(A_i)$ is the number of unassigned tasks then

$$LB2_i = \lceil \#(A_i)/\lfloor C/t_{min} \rfloor \rceil$$

Lower Bound 3:

All tasks in A_i with task processing times exceeding half the cycle time must be assigned to a different station:

$$LB3_i = \#(x \in A_i \mid C/2 < t_x).$$

Lower Bound 4:

$$LB4_i = \#(x \in A_i \mid 2C/3 < t_x) + 0.5\#(x \in A_i \mid C/3 < t_x \leq 2C/3)$$

The lower bound for node i is $LB_i = \max(LB1_i, LB2_i, LB3_i, LB4_i)$.

2.3 Bounding

- Fathom any node, i, where $LB_i \geq U_{current}$.
- If $U_{current} - LB_i > 1$, U_i, an upper bound on the number of stations in the solution that contains node i; is calculated by using the single pass, maximum time heuristic, H, to assign all unassigned tasks to stations.

$U_i = D_i + H(A_i)$. If $U_i < U_{current}$, set $U_{current} = U_i$.

2.4 Branching

Branching occurs as follows. Branch on the deepest node (that is, the node i with the largest D_i). In case of a tie select the node with the minimum I_i/D_i (that is, the node in the path with the lowest accumulated idle time per station). If there is still a tie select the first node that was checked.

2.5 Stopping Rule

Stop when there is no unfathomed node i with $LB_i < U_{current}$, meaning that no further improvement is possible. The optimal solution is the current solution and it requires $U_{current}$ stations.

Example 1

Consider the problem shown in Figure 3. A summary of the application of the breadth-first algorithm to this problem is given in Figure 4. Using the maximum task time

Figure 3

Precedence Relations for Example 1

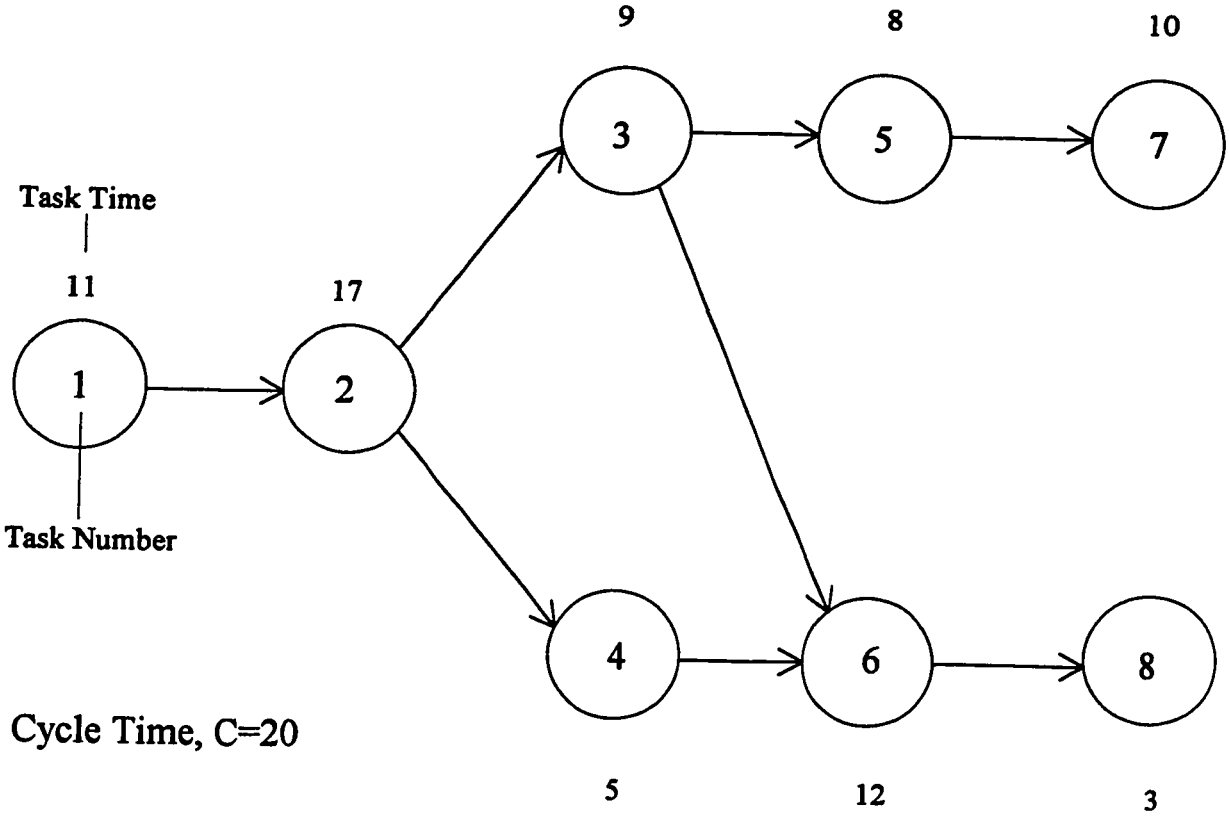
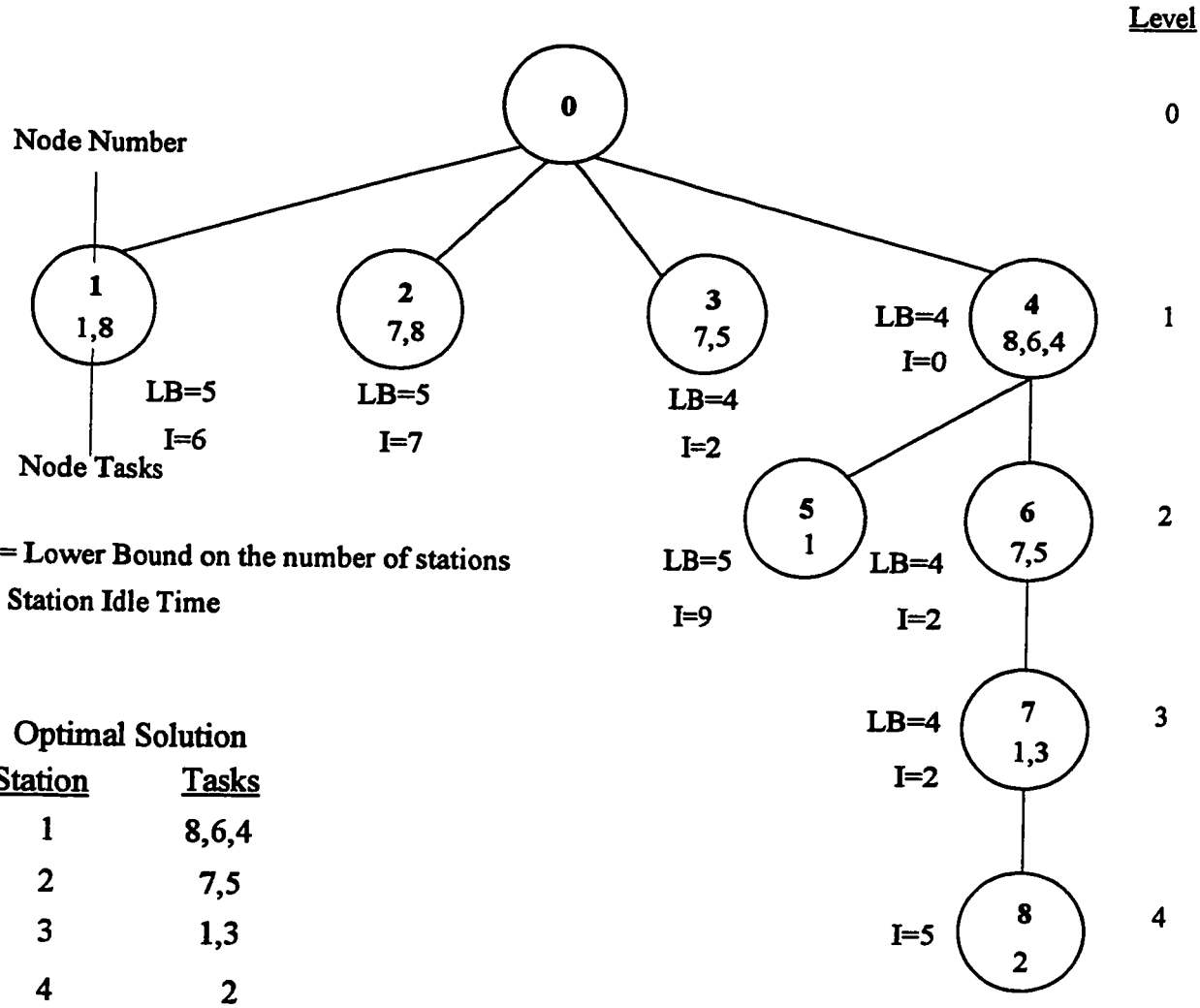


Figure 4

Illustrative Example of the Branch and Bound Algorithm



LB = Lower Bound on the number of stations

I = Station Idle Time

Optimal Solution

Station	Tasks
1	8,6,4
2	7,5
3	1,3
4	2

heuristic, the initial upper bound is found to be 5 stations. This is the current best solution, $U_{current} = 5$. The lower bound is $L^* = \lceil \sum t_i / C \rceil = \lceil 75/20 \rceil = 4$ stations, which is less than $U_{current}$ and so the algorithm moves to Step 2.

Four possible initial nodes are enumerated. Each represents an assignment of tasks to the first station. Node 1, with tasks {1,8}, has a total task time of 14 minutes and an idle time of 6 minutes. The lower bound at this node is $LB_1 = \max (5, 3, 4, 4) = 5$ stations. That is, any solution that contains node 1--tasks {1,8} assigned to station 1--would require at least 5 stations. Because node 1 does not improve on the current best solution it is fathomed. Node 2 also has a lower bound of 5 stations and is fathomed. Nodes 3 and 4 have lower bounds of 4 stations and so may lead to a solution that is better than the current best solution. Consequently, they are considered for branching. As node 4 has less idle time than node 3, it is the node from which branching occurs. Only two nodes follow from node 4. These are node 5 (task {1} assigned to station 2) and node 6 (tasks {7,5} assigned to station 2). The lower bound on node 5 is 5 stations and so node 5 is fathomed. This leaves nodes 3 and 6 for further branching. Node 6 is at a greater depth than node 3 and so is selected for branching. This process continues as shown in Figure 4. The optimal solution consists of nodes 4, 6, 7 and 8. The optimal assignment of tasks to stations is shown in the figure.

4.2. Depth-first Solution Algorithm

Depth-first algorithms for the SALB problem construct a solution one branch at a time, retaining only the current branch in memory. For this reason, they offer a

significant advantage over other algorithms in terms of storage space required. Depth-first algorithms employ a variety of strategies to avoid complete enumeration of all possible sequences of stations. Not all of these strategies are as successful for the SULB problem. In Talbot and Patterson's [1986] algorithm for the SALB problem, upper and lower bounds on the station number to which each task can be assigned and still improve the current best solution are computed. Tasks are assigned to stations in such a way that these bounds are not crossed. In the SULB problem stations are not created sequentially from the beginning to the end of the line. The last station can occur at any point along the U-line making Talbot and Patterson's bounds very loose when applied to the SULB problem.

The Fable and Eureka depth-first algorithms of Johnson [1988] and Hoffmann [1992], respectively, may be applied to the SULB problem. The most successful algorithm we tested was a combination of the two with some modifications. In the SALB problem, depth-first B&B is facilitated by a numbering scheme in which if $(x,y)\in P$, then $x<y$. Thus, if station i contains task x and station j contains task y then $i\leq j$. This property is not found in the SULB problem. The search for tasks to be added to a station after a task x is assigned, proceeds in both directions from task x in the precedence graph. Consequently the depth-first algorithm acts on an ordered list of available tasks similar to the PEND list of Van Assche and Herroelen [1978] (appendix 3) instead of a list of tasks ordered by task number. While this eliminates considering the same station assignment twice at any branching point, it does not eliminate considering the duplicate subsets with more than one station.

The procedure for implementing the depth-first algorithm is similar to that used in

the breadth-first algorithm. Initially a heuristic upper bound is found, $U_H=H(x|x\epsilon S)$, and compared to a lower bound, $L^* = \lceil \sum_{x\epsilon S} t_x/C \rceil$, (steps 1.1 and 1.2 in the breadth-first algorithm). If the upper bound exceeds the lower bound, $U_H > L^*$, the depth-first begins. As in Eureka an initial current slack is defined, $CS = L^* - \sum_{x\epsilon S} t_x$. The algorithm builds a solution along a single branch until the accumulated idle time exceeds the current slack, at which point backtracking occurs. If no solution can be found with the current slack, the slack is increased by one cycle time, CS←CS+C, and the procedure recommences, provided the new slack does not correspond to the heuristic upper bound. The explicit search of the solution tree is further reduced by incorporating the following rules into the algorithm. 1) If the label of a newly created station equals the label of a previous station then backtracking occurs (Johnson [1988]). 2) A lower bound for station i, $LB_i = \max(LB1_i, LB2_i, LB3_i, LB4_i)$ (step 2.2 in the breadth-first algorithm) on the number of stations required to complete the remaining unassigned tasks is computed. If it exceeds the upper bound corresponding to the current slack then backtracking occurs.

5. Empirical Results

An empirical study was undertaken to examine the SULB problem for the purpose of gaining insights into the nature of the SULB problem and identifying the most efficient optimal solution algorithms.

In the first part of the study, the efficiencies of the U-line and the straight-line were compared (efficiency was defined to be the number of stations in the line). Then the recursive fixing and reaching DP approaches were compared for the two problems. Next

the computation time and storage space requirements of the reaching DP algorithm, and the breadth-first and depth-first B&B algorithms were investigated for the SULB problem.

Thirty small problems, ranging in size from 7 to 18 tasks, were selected. Five were from the SALB literature (Bowman [1960], Dar-El [1964], Jackson [1956], Jaeschke [1964] and Merten [1967]) and 25 were randomly generated (Problem Set 1 in Appendix 4). Six different randomly generated cycle times, ranging from $C=\max\{t_x\}$ to $C=2\max\{t_x\}$, were selected for each problem. The resulting $6\times30=180$ instances were solved first as SALB instances and then as SULB instances. In 87 SALB instances and 134 SULB instances, heuristic H gave a solution where the number of stations equalled the theoretical lower bound, $\lceil\sum t_x/C\rceil$. For the remaining 46 "difficult" SULB instances, optimal solutions were obtained using recursive fixing DP, reaching DP, breadth-first B&B and depth-first B&B algorithms. The algorithms were coded in QuickBasic and run on a 486, 50 mhz computer. The results, from which the following insights are gleaned, are shown in Table 1.

1. The straight-line is a special case of the U-line. For a significant number of instances, a straight-line is less efficient than a U-line. In 22 percent of the instances studied the straight-line required more stations than the U-line, and straight lines required an average of 4.3 % more stations than the U-lines.
2. Heuristic H was more effective for solving the SULB problem than for the SALB problem. It gave an optimal solution in 81 percent of the SULB instances and 62 percent of the SALB instances. The SULB problem is less constrained than the SALB problem, and so it is easier for a heuristic to find very good assignments

Table 1

Results from Part 1 of the Empirical Study of the SULB Problem

Section 1: A Comparison of SALB and SULB Optimal Solutions

	Straight Line	U-Line
Percent of problems where fewer stations were required	0 %	22%
Percent of problems where the solution from the *maximum task time* heuristic equals the theoretical lower bound	48%	74%
Percent of problems where optimal solution was obtained from the *maximum task time* heuristic	62%	81%

Section 2: A Comparison of Computational Requirements for the SALB and SULB Problems when the Reaching Dynamic Programming Algorithm was Used

	Straight Line	U-Line
Average number of feasible subsets enumerated	55	599
Average time required to generate feasible subsets	6 sec.	129 sec.

Section 3: An Analysis of Bounds on the Number of Feasible Subsets

	Number of subsets	Bound for the Straight-Line $\#(S^f)/\{\sum_z L_z\}$	Actual Number of Subsets / Bound		
			$\#(S)/\{[(\sum_z L_z)^2 + \sum_z L_z]/2\}$	$\#(S)/\{[((N+w)/w)^{2w} + ((N+w)/w)^w]/2\}$	$\#(S)/\{[\#(S^f)^2+\#(S^f)]/2\}$
Straight-Line	$\#(S^f)$ subsets	45%	—	—	—
U-Line	#(S) subsets	—	14.7%	6.0%	41.2%

Section 4: Comparison of Optimal Solution Algorithms for the SULB Problem[1]

Optimal Solution Algorithm	Total Solution Time (seconds)	Total # of Nodes
Dynamic programming (reaching method)	3814	N/A
Breadth-First Branch and Bound		
Algorithm 1 Branch on node i with the minimum I_i/D_i. Break ties by selecting i with the maximum D_i. Use lower bound $LB1_i$. Do not use label dominance rule.	1302	3530
Algorithm 2 Same as Algorithm 1 except that label dominance rule is used.	522	1085
Algorithm 3 Branch on node i with the maximum level D_i until first solution is found. Break ties by selecting i with minimum I_i/D_i. Use lower bound $LB1_i$ and label dominance rule. When the first solution is found revert to Algorithm 2.	508	1020
Algorithm 4 Branch on node i with the maximum level D_i. Break ties by selecting i with the minimum I_i/D_i. Use label dominance rule. Use lower bound $LB2_i$.	471	803
Algorithm 5 Same as Algorithm 4 except that the maximum task time heuristic is used to calculate the lower bound at each node.	1181	803
Algorithm 6 Same as Algorithm 4 except that lower bound $LB_i=\max(LB3_i, LB4_i)$ is used.	378	692
Algorithm 7 Same as Algorithm 4 except that lower bound $LB_i=\max(LB1_i, LB2_i, LB3_i, LB4_i)$ is used.	344	556
Depth-First Branch and Bound		
Algorithm 1 A modification of the Eureka implicit enumeration algorithm (Hoffmann [1992]).	2136	N/A
Algorithm 2 Same as Algorithm 1 except that lower bound $LB_i=\max(LB1_i, LB2_i, LB3_i, LB4_i)$ is used.	1952	N/A
Algorithm 3 Same as Algorithm 2 except that label dominance rule is used.	1074	N/A

Note: 1. Solutions for the 46 problems in problem set 1 where the heuristic upper bound does not achieve the theoretical lower bound.

Part 5: Comparison of Optimal Solution Algorithms for the SULB Problem for Larger, More Difficult Problems[2]

Optimal Solution Algorithm	% Proven Optimal	Total # of Stations
Breadth-First B&B		
Problem Set 2	91%	1241
Problem Set 3	70%	1168
Depth-First B&B		
Problem Set 2	75%	1252
Problem Set 3	70%	1158

Note: 2. Solutions for the problems in problem sets 2 and 3.

of tasks to stations. Being greedy or myopic in the assignment of tasks to early stations is less likely to cause difficulty with assignments at later stations.

3. A comparison of the computational requirements for the SALB and SULB problems when the reaching DP algorithm is used, shows how difficult the SULB problem is (section 2 of Table 1). The average number of subsets that are generated was 55 for the SALB problem and 599 for the SULB problem, and the average time to generate all these subsets was 6 seconds for the SALB problem compared to 129 seconds for the SULB problem.

4. Section 3 of Table 1 shows that the Schrage and Baker bound and the Steiner bound on the number of feasible subsets generated by dynamic programming is poor for the SULB problem. The actual number of feasible subsets is only 14.7% of the Schrage and Baker bound and 6.0% of the Steiner bound. This is not surprising since these bounds were not very tight for the simpler SALB problem. Solving a SALB problem to determine $\#(S^f)$, the actual number of forward subsets, and computing $\#(S) \leq [\#(S_f)^2+\#(S_f)]/2$, gives a much better bound for the SULB problem but since the SALB problem is a NP hard problem itself, the usefulness of this bound is doubtful. Determination of tighter upper bounds is an area of on-going research.

5. The reaching DP approach is more effective than the recursive fixing approach, in terms of computation time and storage space. Due to space limitations in the algorithm provided by Miltenburg and Wijngaard [1994] the recursive fixing algorithm could only solve 150 of the 180 instances. For those instances which

both algorithms solved, the reaching approach was faster and created fewer subsets. The computation time for the reaching approach was 43 percent of the time for recursive fixing, and the number of subsets was 46 percent of the number for recursive fixing. This result is consistent with Kao and Queyranne [1982] and Steiner [1990] who also found that reaching outperformed recursive fixing.

6. The total time taken by the best breadth-first B&B algorithm (version 7 in section 4 of Table 1) to solve the 46 difficult SULB instances was 9% of the time taken by the reaching DP algorithm (344 seconds versus 3814 seconds). Most of the time the breadth-first algorithm is able to either find a solution or fathom all the branches, and stop, after considering only a small fraction of the possible branches and nodes. Because the upper and lower bounds are quite tight, the algorithm is able to quickly determine when improvement beyond the current best solution is not possible. DP on the other hand, does not stop until it has completed all its computations. It is clear that the effectiveness of B&B depends on the quality of the upper and lower bounds and that the bounds presented in this paper are of high quality.

7. The best combination of branching and bounding rules for the problems considered here was to branch on the depth, D_i, break ties using average idle time per station, use a lower bound at each node that was the maximum of four lower bounds, and not use an upper bound at each node (version 7 in section 4 of Table 1). The dominance rules are even more important for the SULB problem than for the SALB problem. This is because the potential for duplicating subsets and stations

is much higher on U-lines than on straight lines.

8. Although the best breadth-first algorithm solved the 46 difficult problems in 32% of the time required by the depth-first algorithm, two observations suggested that further investigation into depth-first B&B was warranted. Depth-first took less time than breadth-first in 10 of the 46 problems, and 75% of the total computation time for depth-first was spent on two problems. This suggested that Hoffmann's observation that depth-first algorithms for the SALB problem either find solutions quickly or take a very long time, also applies to the SULB problem.

In the next part of the study, the two most promising B&B algorithms—one breadth-first and one depth-first—were tested on two sets of randomly generated, difficult instances.

Problem Set 2 (see Appendix 4) was similar to Problem Set 1, except that the number of tasks was larger—an average of 15 versus 11 tasks—and the minimum task completion time was constrained to be at least as large as half the maximum completion time. Each problem was tested with six different randomly selected cycle times, uniformly distributed between $\max\{t_x\}$ and $2\max\{t_x\}$, giving a total of $6\times40=240$ instances. Heuristic H achieved the lower bound in 26% of these problems.

Even larger U-lines were considered in Problem Set 3 (see Appendix 4). The number of tasks ranged from 20 to 40 and averaged 30. Again each problem was tested with six randomly selected cycle times, giving a total of $6\times15=90$ instances. Heuristic H achieved the lower bound in 47% of these problems.

A maximum computation time limit of 300 seconds was set for each instance. (As the intent of the study was to compare optimal solution algorithms, the depth-first algorithm does not employ Hoffmann's heuristic to solve the instance once the time limit is exceeded.) The results are summarized in section 5 of Table 1.

9. Neither the best breadth-first or depth-first algorithm was able to find the optimal solution for all instances within 300 seconds.

10. The size of the solution space grows rapidly with the number of tasks, the width of the precedence graph and the number of precedence relationships. The relationship of cycle time to task completion times also affects the performance of the algorithms. The instances which could not be solved within the time limit by the breadth-first algorithm had the following (average) characteristics: size=17.9 tasks, width=4.7 tasks, cycle time/maximum task completion time=1.36, and computation time>300 seconds. These compare to the (average) characteristics for the instances which were solved optimally: size=14.8 tasks, width=3.85 tasks, cycle time/maximum task completion time=1.56, and computation time=52.6 seconds.

11. Breadth-first outperforms depth-first for the smaller instances (Problem Set 2). Depth-first is slightly better than breadth-first for the larger problems (Problem Set 3). As the number of tasks increase, the size of the branch and bound tree grows until the number of successor nodes to any node is so large that the breadth-first algorithm spends most of its time enumerating successor nodes and very little time working through the tree to a solution. Depth-first, on the other hand, works down

a single branch and may at least find a good solution in the time allowed.

12. The smaller instances (Problem Set 2) that breadth-first and depth-first could not solve were identical, with one exception. For the larger instances (Problem Set 3), each was unable to solve 27 instances, with nine of them being different. Consequently, it may a good idea to try both algorithms for certain types of large instances.

6. Summary and Conclusions

The SULB problem differs from the SALB problem in one major way. In the SULB problem, tasks may be assigned to stations while moving through the precedence graph in two directions (forward and backward) simultaneously, while in the SALB problem tasks may be assigned while moving through the graph in one direction (forward or backward) only. Although this causes the complexity of the problem to increase dramatically, it also increases the number of ways tasks can be combined into stations, with the result that better solutions (relative to the SALB problem) often exist. For example, in the empirical study reported here, 22% of 180 small SULB problems required fewer stations than the corresponding SALB problems. The increased complexity of the SULB problem (relative to the SALB problem) has the unfortunate effect that some approaches that work well for the SALB problem--recursive fixing dynamic programming and Talbot and Patterson's depth-first B&B algorithm--both of which take advantage of the simpler structure of the SALB problem do not work well for the SULB problem.

Three new SULB algorithms were presented in this paper--a reaching dynamic

programming algorithm, a breadth-first B&B algorithm and a depth-first B&B algorithm. The complexity of the dynamic programming algorithm is $O(N^2[\#(S^f)]^2)$, compared to $O(N^2[\#(S^f)])$ for the SALB problem. In an empirical study, the reaching dynamic programming algorithm was found to be more than twice as effective as the recursive fixing algorithm. The breadth-first and depth-first B&B algorithms were more efficient than the reaching dynamic programming approach, a result that is consistent with findings reported in the literature for the SALB problem. However there are limitations on the size and difficulty that these B&B algorithms can solve to optimality. The best algorithm was able to prove optimality in 91% of 10-20 task problems and 70% of 20-40 task problems within a 300 second time limit. One finding which will influence future research is that in more difficult problems, where there are potentially hundreds of successor nodes to any node, the time required to enumerate all potential successors is so large that algorithms which require enumeration of most or all potential successors cannot be used.

The U-line balancing problem is a new problem and more research work, including the following, needs to be done to understand it better. Some promising directions for future research are:

1. Better bounds on the number of feasible subsets are needed for the DP algorithm. Work can be done to incorporate bounds into the DP algorithm.
2. More work can be done on the B&B algorithms to investigate the effect of different branching rules, upper and lower bounds, dominance rules, and so on.
3. The SULB model can be broadened to include other considerations such as

    a) concentrate the idle in one station so that improvement efforts can be focussed

at that station (in accordance with JIT practices),

b) minimize the distance travelled by operators, and

c) locate certain tasks or stations in pre-specified locations.

4. Research is needed on multi-product U-lines, multiple U-lines, and U-lines where the task processing times are stochastic.

5. The development of fast effective heuristics for solving U-line balancing problems.

References

Baybars, I. "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem," *Mgmt. Sci.*, 32 (1986), 909-932.

Bowman, E.H., "Assembly Line Balancing by Linear Programming," *Oper. Res.*, 8, 3 (1960), 385-389.

Dar-El, E.M., "Assembly Line Balancing - An Improvement on the Ranked Positional Weight Technique," *Jour. of Industrial Engineering*, 15, 2, (1964), 73-77.

Denardo, E.V. and B.L. Fox, "Shortest-Route Methods: 1. Reaching, Pruning, and Buckets," *Oper. Res.*,27 (1979), 161-186.

Ghosh S. and R.J. Gagnon, "A Comprehensive Literature Review and Analysis of the Design, Balancing and Scheduling of Assembly Systems," *Int. J. Prod. Res.*, 27, 4 (1989) 637-670.

Hackman, S.T., M.J. Magazine and T. S. Wee, "Fast, Effective Algorithms For Simple Assembly Line Balancing Problems," *Oper. Res.*, 37, 6 (1989), 916-924.

Hall, R. W., *Zero Inventories*, Dow Jones-Irwin, Homewood, Illinois (1983).

Hoffmann, T.R., "Eureka: A Hybrid System For Assembly Line Balancing," *Mgmt. Sci.*, 38, 1 (1992), 39-47.

Jackson, J.R., "A Computing Procedure for a Line Balancing Problem," *Mgmt. Sci.*, 2,3 (1956), 261-271.

Jaeschke, G., " Eineallgemaine Methods Zur Losung Kombinatorisher Probleme," Ablaufund Planungforschung, 5, (1964), 133-153.

Japan Management Association, *Canon Production System*, Productivity Press, Portland,

OR, 1987.

Johnson, R.V., "Assembly Line Balancing Algorithms: Computational Comparisons,"*Int. J. Prod. Res.*, 19, (1981), 277-287.

Johnson, R.V., "Optimally Balancing Large Assembly Lines with 'Fable' ", *Mgmt. Sci.*, 34, 1, (1988), 240-253.

Kao, E.P.C. and M. Queyranne, "On Dynamic Programming Methods for Assembly Line Balancing," *Oper. Res.*, 30, (1982), 375-390.

Lawler, E.L. "Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems," *Report BW 106/79, Stichting Mathematisch Centrum*, Amsterdam, (1979).

Merten, P., "Assembly Line Balancing by Partial Enumeration", *Ablaufund Planungforschung*, 8, (1967), 429-433.

Miltenburg G.J. and J. Wijngaard, "The U-Line Balancing Problem," *Man. Sci., 40, 10,* (1994), 1378-1388.

Monden, Y., *Toyota Production System*, Second Edition, Industrial Engineering Press, Institute of Industrial Engineers, Norcross, GA, (1993).

Provan J.S. and M.O. Ball, "The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected," *SIAM J. Comput.*, 12, (1983), 777-788.

Queyranne M., "Bounds for Assembly Line Balancing Heuristics," *Oper. Res.*, 33, (1985), 1353-1359.

Schrage L. and K.R. Baker,"Dynamic Programming Solution of Sequencing Problems with Precedence Constraints," *Oper. Res.*, 26, (1978), 444-449.

Steiner G., "On The Complexity Of Dynamic Programming For Sequencing Problems With Precedence Constraints," *Annals of Oper. Res.*, 26, (1990), 103-123.

Talbot, F.B. and J.H. Patterson, "An Integer Programming Algorithm With Network Cuts for Solving The Assembly Line Balancing Problem," *Mgmt. Sci.*, 30, 1, (1984), 85-99.

Talbot, F.B., Patterson, J.H., and W. V. Gehrlein, "A Comparative Evaluation of Heuristic Line Balancing Techniques," *Mgmt. Sci.*, 32, 4, (1986), 430-454.

Van Assche, F. and W.S. Herreolen, "An Optimal Procedure for the Single-Model Deterministic Assembly Line Balancing Problem," *Eur. J. of O. R.*, 3, (1978), 142-149.

Wee, T.S. and M. J. Magazine, "Assembly Line Balancing As Generalized Bin Packing," *Operations Research Letters*, 1, (1982), 56-58.

Appendix 1

A Reaching Dynamic Programming Example

Consider a problem with the tasks, processing times and precedence constraints shown in Figure 3. Table 2 shows the implementation of the reaching approach for this problem.

First all subsets of size n=1 task are generated. They are {1}, {7} and {8}. The costs are F[{1}] = 0 full stations + 11 time units, F[{7}]=0+10 and F[{8}]=0+3. These subsets are stored in List(1) and are denoted $\{1\}=S_1^{(1)}$, $\{7\}=S_2^{(1)}$ and $\{8\}=S_3^{(1)}$; and $\#(S_i^{(1)})=3$.

Next all subsets of size n=2, $S_{i,j}^{(2)}$, are generated from the subsets of size one, $S_i^{(1)}$, in List(1). For $S_1^{(1)}=\{1\}$ this gives, $S_{1,1}^{(2)}=S_1^{(1)}+\{2\}=\{1,2\}$, $S_{1,2}^{(2)}=S_1^{(1)}+\{7\}=\{1,7\}$ and $S_{1,3}^{(2)}=S_1^{(1)}+\{8\}=\{1,8\}$. The costs of these subsets are $F[S_{1,1}^{(2)}]=F[S_1^{(1)}]+g(\{2\})=1+17$, $F[S_{1,2}^{(2)}]=F[S_1^{(1)}]+g(\{7\})=1+10$, and $F[S_{1,3}^{(2)}]=0+14$. These results are stored in List(a).

When $S_2^{(1)}=\{7\}$, $S_{2,1}^{(2)}=S_2^{(1)}+\{1\}=\{7,1\}$ and $F[S_{2,1}^{(2)}]=F[S_2^{(1)}]+g(\{1\})=1+11$; $S_{2,2}^{(2)}=S_2^{(1)}+\{5\}=\{7,5\}$ and $F[S_{2,2}^{(2)}]=F[S_2^{(1)}]+g(\{5\})=0+18$; and $S_{2,3}^{(2)}=\{7,8\}$ with $F[S_{2,3}^{(2)}]=0+13$. These results are stored in List(b).

List(a) and List(b) are now merged. Two subsets in these lists contain the same tasks, $S_{1,2}^{(2)}=S_{2,1}^{(2)}=\{1,7\}$. Since the cost $F[S_{1,2}^{(2)}] < F[S_{2,1}^{(2)}]$, $S_{1,2}^{(2)}$ is retained and $S_{2,1}^{(2)}$ is dropped. The results of the merge are stored in List(a). The process continues for $S_{3,j}^{(2)}$. $S_{3,j}^{(2)}$, u(j) and $F[S_{3,j}^{(2)}]$ are stored in List(b) which, when completed, is merged with

Table 2

An Illustrative Example of the Reaching DP Algorithm

Size, n	i	$S_i^{(n-1)}$	u	$F[S_i^{(n-1)}]$		$S_i^{(n-1)}+\{1\}$	$S_i^{(n-1)}+\{2\}$	$S_i^{(n-1)}+\{3\}$	$S_i^{(n-1)}+\{4\}$	$S_i^{(n-1)}+\{5\}$	$S_i^{(n-1)}+\{6\}$	$S_i^{(n-1)}+\{7\}$	$S_i^{(n-1)}+\{8\}$
1	1	Ø	–	0	j $S_{1,j}^{(1)}$ $F[S_{1,j}^{(1)}]$	1 {1} 0+11						2 {7} 0+10	3 {8} 0+3
2	1	{1}	1	0+11	j $S_{1,j}^{(2)}$ $F[S_{1,j}^{(2)}]$		1 {1,2} 1+17					2 {1,7} 1+10	3 {1,8} 0+14
	2	{7}	7	0+10	j $S_{2,j}^{(2)}$ $F[S_{2,j}^{(2)}]$	1 {7,1} 1+11				2 {7,5} 0+18			3 {7,8} 0+13
	3	{8}	8	0+3	j $S_{3,j}^{(2)}$ $F[S_{3,j}^{(2)}]$	1 {8,1} 0+14					2 {8,6} 0+15		3 {8,7} 0+13
3	1	{1,2}	2	1+17	j $S_{1,j}^{(3)}$ $F[S_{1,j}^{(3)}]$			1 {1,2,3} 2+9	2 {1,2,4} 2+5			3 {1,2,7} 2+10	4 {1,2,8} 1+20
	2	{1,7}	7	1+10	j $S_{2,j}^{(3)}$ $F[S_{2,j}^{(3)}]$		1 {1,7,2} 2+17			2 {1,7,5} 1+18			3 {1,7,8} 1+13
	3	{1,8}	8	0+14	j $S_{3,j}^{(3)}$ $F[S_{3,j}^{(3)}]$		1 {1,8,2} 1+17					2 {1,8,7} 1+10	
	4	{5,7}	5	0+18	j $S_{4,j}^{(3)}$ $F[S_{4,j}^{(3)}]$	1 {5,7,1} 1+11							2 {7,5,8} 1+3
	5	{6,8}	6	0+15	j $S_{5,j}^{(3)}$ $F[S_{5,j}^{(3)}]$	1 {6,8,1} 1+11			2 {6,8,4} 0+20			3 {6,8,7} 1+10	
	6	{7,8}	7	0+13	j $S_{6,j}^{(3)}$ $F[S_{6,j}^{(3)}]$	1 {7,8,1} 1+11				2 {7,8,5} 1+8	3 {7,8,6} 1+12		
etc.													

List(a). As this completes the generation of subsets of size two, the results are stored in List(n=2).

This process then continues for n=3,4,...,N.

Appendix 2

A Bound on the Number of Feasible Subsets

Define the following notation:

B_i = a backward feasible subset of S.

$S\text{-}B_i$ = the forward feasible subset which is the complement to B_i.

$f(B_i)$ = the number of backward feasible subsets contained in B_i including $\varnothing$.

$f'(S\text{-}B_i)$ = the number of forward feasible subsets contained in $S\text{-}B_i$ including $\varnothing$.

#(S) = the total number of feasible subsets enumerated in the DP algorithm

Consider a problem consisting of the four tasks shown in Figure 5. The backward feasible subsets for this problem are $B_1=\varnothing$, $B_2=\{3\}$, $B_3=\{4\}$, $B_4=\{4,2\}$, $B_5=\{4,3\}$, $B_6=\{4,3,2\}$ and $B_7=\{4,3,2,1\}$, with the total number $f(S)=7$. For each backward feasible subset B_i the complementary subset $S\text{-}B_i$ is a forward feasible subset. For example, $S\text{-}B_4=\{1,3\}$. Thus the total number of forward feasible subsets $f'(S)=f(S)=7$. Recall the recursive fixing approach for generating feasible subsets (section 3.2). Each backward feasible subset, B_i, $i=1,2,\ldots,f(S)$, is combined with every forward feasible subset in $S\text{-}B_i$, including the null subset. Therefore an initial upper bound on the total number of feasible subsets, #(S), will be

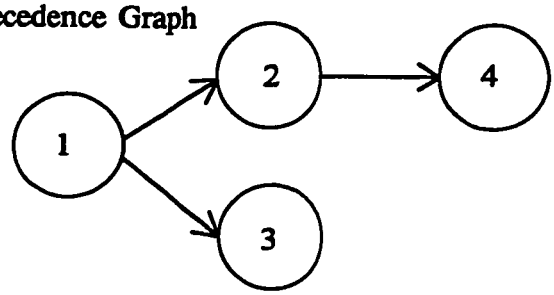
$$\#(S) = \sum_{i=1}^{f(S)} f'(S\text{-}B_i) \le f(S)\times f'(S) = f(S)^2$$

This bound may be reduced further by the following observation. Consider any subset B_i. With the exception of the complete subset B_i, for every backward feasible subset contained

Figure 5

Bound on the Number of Feasible Subsets

Part 1: The Precedence Graph



Part 2: Combining Forward Subsets With Backward Subsets

i	S_i^b	Subsets of $S_i^f = S - S_i^b$, $S_{i,j}^f$, $j=1,2,\ldots,n(S_i^f)$	$\#(S_i^f)$
1	$\varnothing$	$\varnothing,\{1\},\{1,2\},\{1,3\},\{1,2,3\},\{1,2,4\},\{1,2,3,4\}$	7
2	{3}	$\varnothing,\{1\},\{1,2\},\{1,2,4\}$	4
3	{4}	$\varnothing,\{1\},\{1,2\},\{1,3\},\{1,2,3\}$	5
4	{4,2}	$\varnothing,\{1\},\{1,3\}$	3
5	{4,3}	$\varnothing,\{1\},\{1,2\}$	3
6	{4,3,2}	$\varnothing,\{1\}$	2
7	{4,3,2,1}	$\varnothing$	1
$\#(S^b)=7$			$\sum_{i=1}^{n(S^b)} n(S_i^f) = 25$

Part 3: Combining Backward Subsets With Forward Subsets

u	S_u^f	Subsets of $S_u^b = S - S_u^f$, $S_{u,v}^b$, $v=1,2,\ldots,n(S_u^b)$	$\#(S_u^b)$
1	$\varnothing$	$\varnothing,\{4\},\{3\},\{4,3\},\{4,2\},\{4,3,2\},\{4,3,2,1\}$	7
2	{1}	$\varnothing,\{4\},\{3\},\{4,3\},\{4,2\},\{4,3,2\}$	6
3	{1,2}	$\varnothing,\{4\},\{3\},\{4,3\}$	4
4	{1,3}	$\varnothing,\{4\},\{4,2\}$	3
5	{1,2,3}	$\varnothing,\{4\}$	2
6	{1,2,4}	$\varnothing,\{3\}$	2
7	{1,2,3,4}	$\varnothing$	1
$\#(S^f)=7$			$\sum_{u=1}^{n(S^f)} n(S_u^b) = 25$

in b_i there will be one complementary forward feasible subset whose tasks are not all in S-B_i and are thus not feasible. For example, $B_4=\{4,2\}$ contains feasible backward subsets $\{\varnothing\}$, $\{4\}$ as well as $\{4,2\}$ so subsets $\{1,2,3,4\}$ and $\{1,2,3\}$ can no longer be contained in S-B_4. Note also that there may be other forward subsets which are no longer feasible, in this case $\{1,2\}$. Thus for any subset B_i with $f(B_i)$ backward feasible subsets the number of forward feasible subsets in S-B_i is

$$f'(S\text{-}B_i) \le f(S) - f(B_i) + 1.$$

This leads to the following lemma.

Lemma: An upper bound on the number of feasible subsets, #(S), for the SULB problem is $\#(S) \le [f(S)^2+f(S)]/2$.

Proof:

$$\#(S) = \sum_{i=1}^{f(S)} f'(S\text{-}B_i)$$

$$\rightarrow \#(S) \le \sum_{i=1}^{f(S)} [f(S) - f(B_i) + 1]$$

$$\rightarrow \#(S) \le \sum_{i=1}^{f(S)} [f(S) - f(B_i) + 1]$$

$$\rightarrow \#(S) \le \sum_{i=1}^{f(S)} f(S) - \sum_{i=1}^{f(S)} f(B_i) + f(S)$$

$$\rightarrow \#(S) + \sum_{i=1}^{f(S)} f(B_i) \le \sum_{i=1}^{f(S)} f(S) + f(S)$$

$$\rightarrow \#(S) + \#(S) \le f(S)^2 + f(S) \qquad \text{because} \quad \sum_{i=1}^{f(S)} f(B_i) = \#(S)$$

$$\rightarrow \#(S) \le [f(S)^2 + f(S)]/2$$

In Section 3.4 the number of forward feasible subsets in S has been denoted as $\#(S^f)$. The

upper bound on the number of feasible subsets which must be enumerated in a dynamic programming algorithm may be restated as $\#(S) \leq [\#(S^f)^2 + \#(S^f)]/2$.

Appendix 3

Procedure for Enumerating Successor Nodes

The following procedure builds on Van Assche and Herreolen's [1978] work for the SALB problem. From the set of unassigned tasks, A, make a list, L, of tasks available to be assigned to the next node (station) following a node, i. These tasks are those whose predecessors or successors have already been assigned to nodes 1,2,...,i. Place a pointer ahead of the first task on the list.

- Search from task 1 to N create the initial list of available tasks, L. Set Idle=C.

i Loop: *(Successor node enumeration loop)*

- Set i=i+1.

t Loop: *(Task loop)*

- If Idle=0 then exit t loop.
- Starting at the first task past the pointer, move down the list, L, to the first task x with $t_x \leq$ Idle and assign this task to station i. Move the pointer to this task. If no such task can be found exit t loop.
- Idle = Idle - t_x
- Identify and add to the end of the list the predecessors or successors of task x, not already in L, whose predecessors or successors are all already on the list or assigned to previous stations.

Loop to t Loop.

- If the subset of tasks assigned to node i is a subset of a previously enumerated node in this group of successor nodes, set and i = i-1 and exit to the b loop. (That is, do not create this new node.)
- The tasks assigned to node i form a station with idle time I_i = Idle

b Loop: *(Backtracking loop)*

- Remove the task, x, at the pointer from the node i.
- If the task at the pointer has already been removed by a previous backtrack then move back in the list to the last task assigned to the node that has not been removed. Move the pointer to this task, denote it as x, and remove x from the node.
- Idle = Idle + t_x
- Check tasks from the pointer to the end of L which are successors or predecessors of the task removed to determine if they are still available. If they are not remove them from the list.
- If backtracking removes the last task assigned to the node and this task is at the end of the list of assignable tasks then the process of enumerating successor nodes is complete.
- Exit the b loop and return to the i loop.

End of b Loop.

End of i Loop.

Example 2

Recall the problem shown in Figure 3. Table 3 shows the enumeration of the nodes that follow the initial node, node 0, each of which represents a possible station 1 assignment.

Table 3

Enumerating Nodes for the Example in Figure 3

Node	A	Node i following node 0	L	Pointer '	Idle Time	Tasks
Node 0	{1,2,...,8}	i=1	1,7,8	'1,7,8	20	--
			1,7,8	1',7,8	20-11=9	1
			1,7,8,2	1,7,8',2	9-3=6	1,8
			1,7,8,2,6	1,7,8',2,6	no x with $t_x \leq 6$	**1,8 Node 1**
		i=2	**Backtrack**			
			Drop 8; 1,7,-,2	1,7,-',2	9	1
			1,7,-,2	1,7,-',2	no x with $t_x \leq 9$	1; Subset of node 1
			Backtrack			
			Drop 1; -,7,8	-',7,8	20	--
			-,7,8	-,7',8	20-10=10	7
			-,7,8,5	-,7,8',5	10-3=7	7,8
			-,7,8,5,6	-,7,8',5,6	no x with $t_x \leq 7$	**7,8 Node 2**
		i=3	**Backtrack**			
			Drop 8; 1,7,-,5	1,7,-',5	10	7
			1,7,-,5	1,7,-,5'	10-8=2	7,5
			1,7,-,5,3	1,7,-,5',3	no x with $t_x \leq 2$	**7,5 Node 3**
		i=4	**Backtrack**			--
			Drop 5; 1,7,8,-	1,7,8,-'	10	7
			1,7,8,-	1,7,8,-	no x with $t_x \leq 10$	7; Subset of node 2
			Backtrack			
			Drop 7; 1,-,8	1,-',8	20	
			1,-,8	1,-,8'	20-3=17	8
			1,-,8,6	1,-,8,6'	17-12=5	8,6
			1,-,8,6,4	1,-,8,6,4'	5-5=0	**8,6,4 Node 4**
		i=5	**Backtrack**			
			Drop 4; 1,7,8,6,-	1,7,8,6,-'	5	8,6
			1,7,8,6,-	1,7,8,6,-'	no x with $t_x \leq 5$	8,6; Subset of node 4
			Backtrack			
			Drop 6; 1,7,8,-	1,7,8,-'	17	8
			1,7,8,-	1,7,8,-'	no x with $t_x \leq 17$	8; Subset of node 4
			Backtrack			
			Drop 8; 1,7,-	1,7,-'	Enumeration is complete.	

Appendix 4

Procedure For Randomly Generating Line Balancing Problems

Complete the following steps for each problem in problem sets 1, 2 and 3 using the parameters specified in Table 4.

1. Randomly generate the number of tasks, N.
2. Randomly generate the maximum processing time for a task, t_{max}.
3. Randomly generate the individual task processing times, t_x, $i=1,2,\ldots,N$.
4. Randomly generate s_{max}, the maximum number of immediate successors for any task.
5. Use the following procedure to randomly generate the successors for each task $x=1,2,\ldots,N$.
   - 5.1 Randomly generate the number of immediate successors for task x, s_x, from a uniform distribution between 1 and s_{max}.
   - 5.2 The possible immediate successors are $x+1,x+2,\ldots,s_x$. Let the probability that each is a successor be s_x/s_{max}. Randomly select the immediate successors.
6. Check the precedence graph for redundant paths. A redundant path occurs when $(x,y)\in P$, that is y is an immediate successor of x, and another path from x to y already exists in the precedence graph. When a redundant path is found the precedence relation (x,y) may be eliminated according to the rule specified in Table 4.

Some characteristics of the problem sets generated by this procedure are listed in Table 5. Note that the problem set one includes five problems from the literature.

Table 4

Random Problem Generation Parameters

Parameter	Problem Set 1	Problem Set 2	Problem Set 3
Number of problems	30	40	15
Number of tasks, N	u.d.(7-18)[1]	u.d.(10-20)	u.d.(20-40)
Maximum task time, t_{max}	u.d.(10-35)	u.d.(10-35)	u.d.(10-35)
Task processing time, t_x	u.d.(1 to t_{max})	u.d.($.5t_{max}$-t_{max})	u.d.(1-t_{max})
Maximum number of successors, s_{max}	u.d.(3-7)	u.d.(3-7)	u.d.(3-7)
Rule for eliminating redundant paths	Eliminate later relation	Eliminate earlier or later relation (each with a probability of .5)	Eliminate earlier or later relation (each with a probability of .5)

Note: 1. A uniform distribution over an interval from 7 to 18 tasks.

Table 5

Problem Set Characteristics

Characteristic	Problem Set 1	Problem Set 2	Problem Set 3
Number of Tasks: Average, minimum, maximum	11.1, 7, 18	15.1, 10, 20	30.3, 20, 40
Number of Precedence Relations: Average, minimum, maximum	13.8, 6, 20	22.0, 11, 35	42.1, 30, 58
Width of Precedence Graph: Average, minimum, maximum	3.5, 2, 5	3.9, 2, 6	4.2, 3, 6
Density of Precedence Graph: Average, minimum, maximum	.26, .17, .48	.21, .12, .33	.10, .07, .16

EXTENDING THE U-LINE BALANCING PROBLEM: TRAVEL DISTANCES AND BALANCING TWO U-LINES

Abstract

U-shaped production lines are an important part of modern production systems. This paper extends the literature on U-lines in two ways. First, the simple U-line balancing problem is extended to include consideration of operator travel. The total distance traveled by operators in U-lines depends on the line balance. This is different from the traditional line where the total travel distance is independent of the balance, and hence is not considered when developing a balance. Different alternatives for incorporating travel are evaluated. Second, the multiple U-line balancing problem, wherein stations can include tasks from more than one U-line, is introduced through the problem of balancing two U-lines simultaneously. In facilities where U-lines are located in close proximity an opportunity exists to balance two or more U-lines together and reduce the total number of stations required in the facility. Models are formulated for these problems, and branch and bound algorithms are developed to determine optimal line balances. Some computational results are reported.

This article has undergone two revisions for the Journal of Naval Logistics Research and has been submitted for review.

1. Introduction

Just-In-Time (JIT) production methods were introduced in Japan as a means of reducing production costs. By lowering inventory levels, JIT identifies production inefficiencies that can be targeted for improvement. An important feature of JIT is its flexibility to adjust to changes in demand. This is called *shojinka*, and three factors are necessary for achieving it (Monden, [p.160, 1993]):

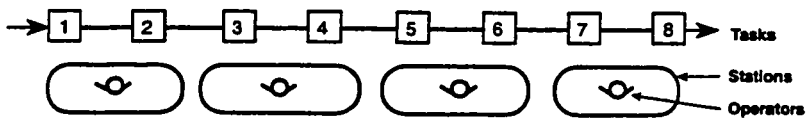
1. "Proper design of machinery layout",
2. "A Multi-function worker", and
3. "Continuous evaluation and ... revisions of the standard operations".

U-shaped production lines are an important part of the first factor. In addition to improving visibility and communication between operators, U-lines can reduce the number of operators required in a facility. Because of the proximity of one side of the U-line to the other, stations may include tasks on both sides of the U-line. This increases the number of possible assignments of tasks to stations, with the result that the number of stations and, consequently, the number of operators needed to operate a U-line will be less than or equal to the number required for the corresponding traditional straight line (Figure 1). Some facilities position U-lines in close proximity and link them together with stations that cross neighbouring lines to provide more opportunities to reduce the number of stations and operators that are needed (Figure 2).

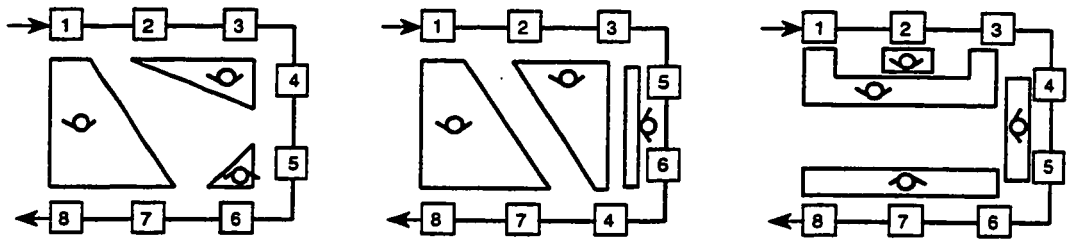
An extensive literature exists on balancing traditional straight lines. See, for example, Talbot and Patterson [1984, 1986], Baybars [1986], Johnson [1981, 1988a, 1988b], Hackman, Magazine and Wee [1989], Ghosh and Gagnon [1989] and Hoffmann

Figure 1

Traditional Line and U-Lines



1. Traditional Line



A. Desirable balance *B. Undesirable balance* *C. Undesirable balance*

2. U-Lines

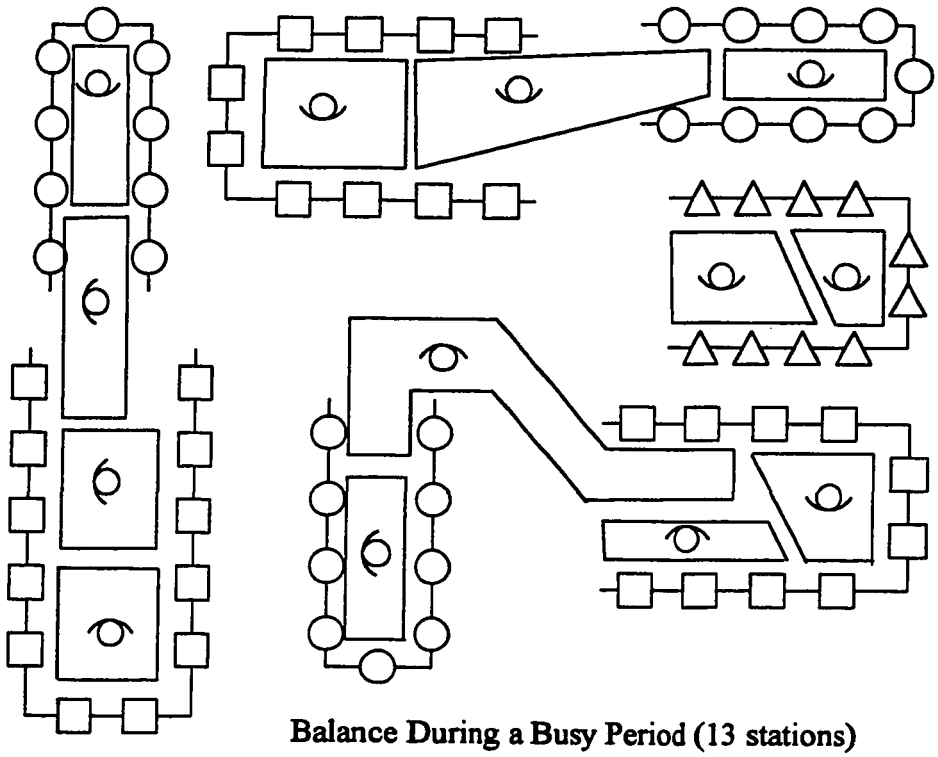
Task	Distance
1	6
2	7
3	4
4	5
5	5
6	5
7	8
8	4

Line	Total distance
Traditional	88
U-line A	84.8
U-line B	97.3

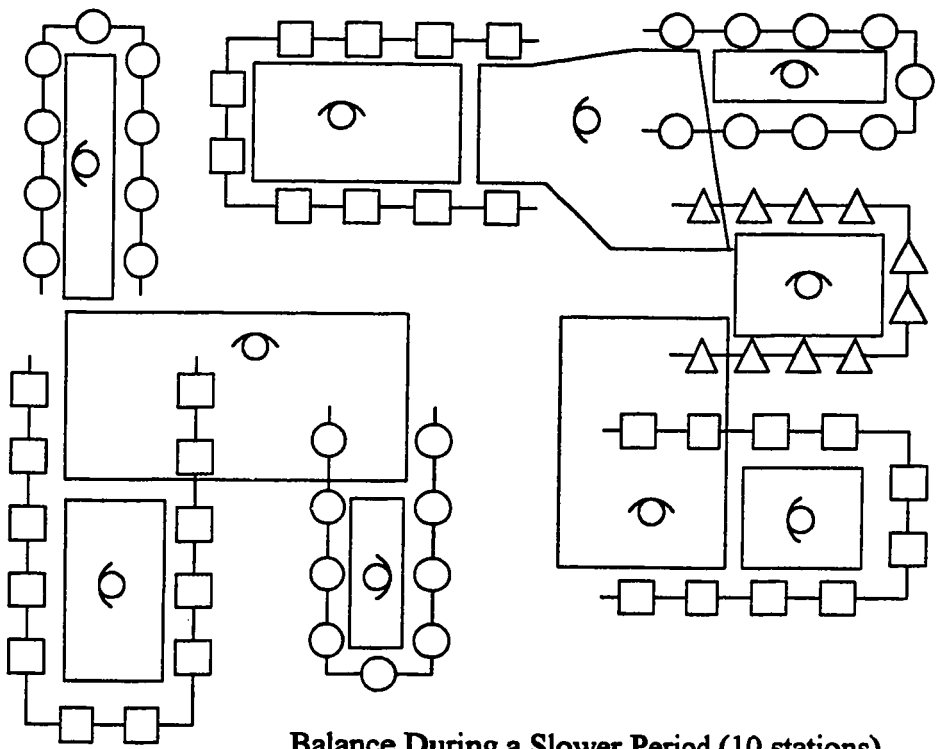
Figure 1. Traditional Line and U-Lines

Figure 2

An Instance of a Multiple U-Line Balancing Problem



Balance During a Busy Period (13 stations)



Balance During a Slower Period (10 stations)

[1992]. The problem of assigning tasks to stations on U-shaped assembly lines, the U-line balancing problem, was introduced by Monden [ch. 11, 1993]. See also Hall [ch. 6, 1983] and the Japan Management Association [pp. 122-125, 1987]. Miltenburg and Wijngaard [1994] formulated a model for the simple U-line balancing (SULB) problem and developed a recursive fixing dynamic programming algorithm to obtain optimal balances for small instances of the problem. More efficient optimal solution algorithms for the U-line balancing problem were presented in Part I of this thesis and an effective branch and bound algorithm for solving SULB problems of typical size (5-20 tasks) was identified.

The SULB problem is the U-line equivalent of the Simple Assembly Line Balancing (SALB-I) problem. While the problems bear many similarities, they differ in several respects because of the difference in line shape and the way the lines are utilized in production. This paper extends the investigation of U-line balancing by considering two such differences: operator travel and the simultaneous balancing of multiple U-lines.

Line balancing algorithms in the literature do not explicitly consider station travel or travel time. This is reasonable on traditional lines for two reasons.

1. An operator on a traditional line starts at the beginning of the station, completes all tasks in the station, and returns to the beginning of the station, all during one cycle time. The distance traveled is twice the sum of the task distances, regardless of where the station is located on the line. (See part 1 of Figure 1.)
2. The total distance traveled by all operators on a traditional line is a constant, equal to twice the sum of all the task distances. As this constant does not depend on how

the line is balanced, travel distance need not be considered in the SALB problem, other than to assume that travel time is part of task completion time.

However, the U-line is different.

1. Travel distance is not a constant. It depends on how tasks are assigned to stations. U-lines A and B in Figure 1 have total travel distances of 89.6 and 99.4 feet respectively because the assignment of tasks to stations is different.
2. The travel distance for a station depends on the shape of the U-line and the location of the station. For example, the last station on U-lines A and B in Figure 1 each contain tasks 5 and 6. However, the time required in these stations is not the same because the travel distances, and hence travel times, are different.
3. In facilities where a number of U-lines are located in close proximity, it is possible to link U-lines with multi-line stations that contain tasks from two or more U-lines. In such cases, account must be taken of the time required to travel between U-lines (Figure 2).
4. The shape of a U-line imposes restrictions on the shape and location of stations. In order to prevent stations from interfering with each other and avoid stations that sprawl and require excessive amounts of travel, location constraints must be added to the model. These are in addition to the cycle time and precedence constraints of the SULB problem. U-line C in Figure 1 is an example of the type of balance that is avoided when location constraints are used and station travel is monitored.

At the Toyota Motor Corporation in Cambridge, Ontario 30-40% of the processes operate on U-lines. During line balancing all task times are considered independent of

travel. As tasks are assigned to stations an allowance is made for time to travel to the next task and to return at the end of the cycle. This time can be a significant portion of total station time. Monden gives a detailed operations routine sheet for a station with 11 tasks and a cycle time of two minutes, where total travel is 22 seconds or 18.3% of the cycle time (p.150, Monden [1993]). Travel is regarded as a non value-added component of production. Its measurement is vital to the activities of the Kaizen group who's goal is to minimize the non value-added components of production. Calculation of travel is also used to limit the maximum travel in any station (In the Cambridge facility, for example, for a 2.3 minute cycle time there is a limit of 5.4 metres of line travel plus travel for parts pickup and returning to the beginning of the station).

Some facilities position U-lines in close proximity and link them together with stations that cross to neighbouring lines to provide more opportunities to reduce the number of stations and operators that are needed. Figure 2 is a good illustration of the multiple U-line balancing problem. Another may be found in Monden [pp. 164-166, 1993]. Schonberger [pp. 140-141, 1982] was first to notice the preference among Japanese manufacturers for multiple U-lines (which he calls U-shaped or parallel-configured lines) with stations that span more than one U-line. "If the natural number of stations for a reasonable balance is, say, two or three, Toyota's preference is to find another two-station or three-station line segment to locate parallel to the first. The parallel lines offer more staffing options." Wantuck [pp.132-162, 1989] gives short case studies of multiple U-lines (which he calls JIT group technology cells) at Ferro Manufacturing, Johnson Controls (especially Figure 7-4), Fisher Guide, Miller Electric Manufacturing

(especially Figure 7-12), Jidosha Kiki Corporation, and Dayco Corporation.

This paper extends previous work on the U-line balancing problem in two ways.

1. The simple U-line balancing problem is extended to include consideration of distances traveled by operators and location restrictions imposed by U-line shape. An optimal branch and bound algorithm for the simple U-line balancing problem with travel is presented. The necessity for explicit consideration of station travel is demonstrated through a computational study. As well as giving better balances for single U-lines, the model provides a framework for considering travel in more complex multiple U-line balancing problems.
2. The simplest multiple U-line balancing problem, the problem of balancing two U-lines simultaneously, is introduced. As we will see, this is the only multiple U-line balancing problem for which optimal balances can be obtained routinely.

The remainder of the paper is organized as follows. A model and an optimal solution algorithm for the simple U-line balancing problem with travel are presented in section 2. In section 3 the two U-line balancing problem with travel is introduced and an optimal solution algorithm is presented. Computational results in section 4 provide justification for explicitly considering travel in U-lines and give insight into the two U-line balancing problem. A discussion follows in section 5, and an illustrative example is worked in the appendix.

2. The Simple U-line Balancing Problem with Travel

2.1 Definitions and Notation

Consider the U-line in Figure 3. An X-Y grid is used to describe the line. Although U-lines may take on a variety of shapes, we assume, for ease of exposition, that U-lines have the general form shown in the figure. A U-line consists of a ***front*** and ***back***, each with identical lengths, L, and a ***side*** of length W. The line starts at point (0,W) and ends at point (0,0). It is relatively straight forward to consider U-lines with different shapes by simply using grid measurements. Changing the shape of the line does not significantly alter the problem but may require additional location constraints.

Define the following notation with tasks denoted by the subscript x and stations by the subscript m.

N = The number of tasks to be completed on the U-line.

$S = \{x \mid x = 1,\ldots,N\}$, the complete set of tasks.

$P = \{(x,y) \mid x,y \in S$, task x is an immediate predecessor of task y$\}$, the set of precedence constraints for the tasks in S.

C = The cycle time.

t_x = The completion time for task x.

L = The length of each of the front and back of the U-line.

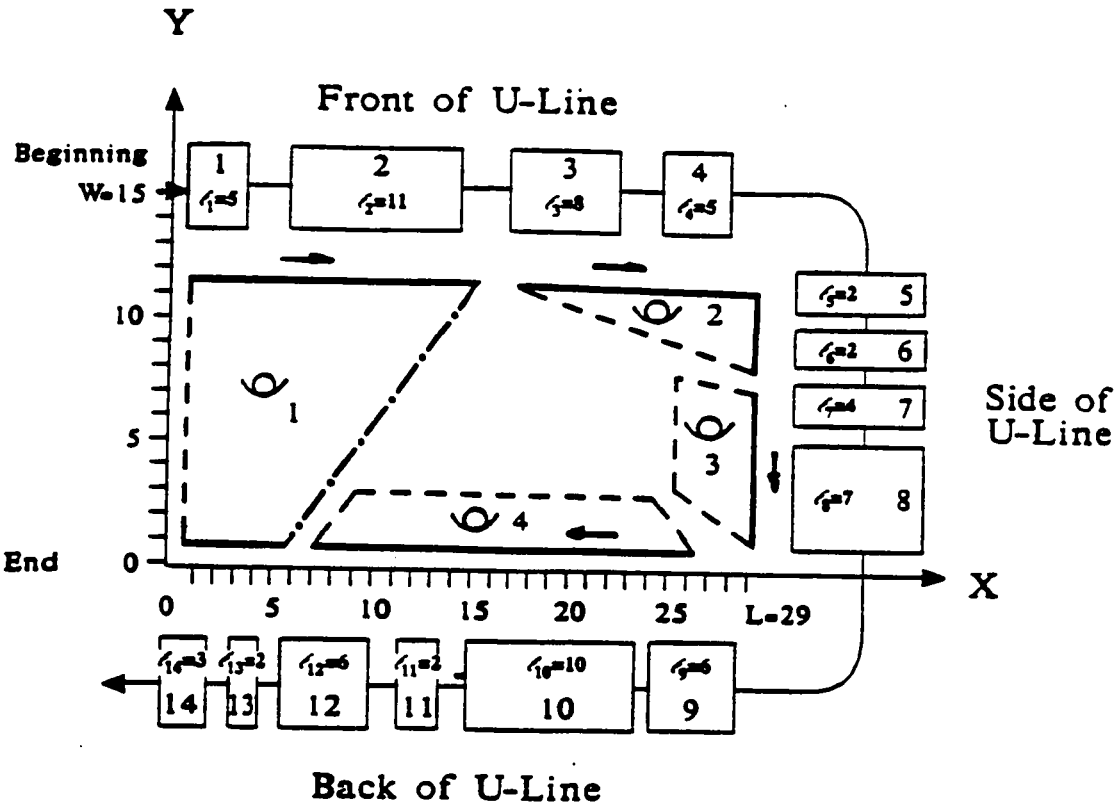
W = The width of the U-line.

ℓ_x = Task distance. The distance on the U-line required by task x.

g = Travel time per unit of travel distance.

Figure 3

U-Line with Travel Distance: Notation and Terminology



Key:

Task distance:	———
Crossover distance:	—·—·—·—
Return distance	– – – –
Regular stations:	2, 3, 4
Crossover station:	1
Operator:	
Task:	□

$0 \le \alpha_x \le 1$ The proportion of task distance not accounted for in task completion time. Hence task travel time is $g\alpha_x\ell_x$.

ST_m = {x | task x is performed at station m}, the set of tasks assigned to station m.

M = The number of stations.

DT_m = The total travel distance in station m.

$g \times DT_m$ = The total travel time in station m.

We assume without loss of generality that the line is designed in such a way that $\sum_{x \in S} \ell_x = 2L + W$.

As discussed in Miltenburg and Wijngaard [1994], tasks may be assigned to stations by working through the precedence graph, P, in either a forward or backward direction or in both directions at the same time. Tasks assigned to stations by working through P in a forward direction are located in sequence along the front of the line. Such tasks are termed ***forward tasks***. Tasks assigned by working through P in a backward direction are located along the back of the line and are termed ***backward tasks***.

Two types of stations are used on simple U-lines, regular stations and crossover stations. A ***regular station*** is one whose tasks are arranged sequentially along the line. Regular stations include either forward or backward tasks, but not both. Stations 2, 3 and 4 in Figure 3 are regular stations. A ***crossover station*** includes forward and backward tasks; that is, tasks located on both the front and back of the U-line. Station 1 in Figure 3 is a cross-over station.

Three types of travel distance are possible in a station, task distance, crossover distance and return distance (Figure 3). ***Task distance***, denoted as ℓ_x, is the distance on

the line required to complete a task. In this analysis, only the travel time not already included in task completion time is considered. Consequently, only $g\alpha_x\ell_x$ is included in the travel time. ***Crossover distance*** only occurs in crossover stations and is the distance traveled when the operator moves from the front of the line to the back. Crossover distance in station m is denoted d_m^c. ***Return distance*** is the distance the operator travels at the end of a cycle returning to the beginning of the station. Return distance is denoted as d_m^r. The total travel distance in station m is

$DT_m = \sum_{x \in ST_m} \ell_x + d_m^c + d_m^r$ and the total travel time is

$g(DT_m) = g\,[\sum_{x \in ST_m} \alpha_x\ell_x + d_m^c + d_m^r]$. At each station the operator must have sufficient complete the tasks assigned to the station and travel the total travel distance.

$$\sum_{x \in ST_m} t_x + g \times DT_m \le C, \quad m=1,2,\ldots,M.$$

2.2 Calculating Travel Distance and Travel Time

The locations of the tasks, and the travel, crossover and return distances are simple functions of the assignments of tasks to stations and the shape of the U-line. Consider the U-line in Figure 3. Task length is shown inside the task box. The front and back of the U-line are each 29 feet long and the U-line is 15 feet wide. Now consider station 1. The operator begins at location (0,15) and completes tasks 1 and 2 moving to location (16,15). He/she crosses to the back of the line to the beginning of task 13 at (5,0) and completes tasks 13 and 14 ending at (0,0). Before the next cycle starts, he/she must return to the beginning of the U-line at (0,15). Therefore the travel distance in station 1 is

$$DT_1 = \sum_{x \in ST_1} \ell_x + d_1^c + d_1^r$$

$= (5+11+2+3) + [(16-5)^2+(15-0)^2]^{0.5} + [(15-0)^2+(0-0)^2]^{0.5}$

$= 21 + 18.6 + 15 = 54.6$ feet

If $\alpha_x=1$ for each task and $g=.3$ second/foot, then the travel time is $g \times DT_1 = .3[1 \times 21 + 18.6 + 15] = 16.4$ seconds. If the cycle time is $C=120$ seconds, then an operator spends $16.4/120 = 14\%$ of the cycle time traveling.

2.3 A Model for the SULB-T Problem

The simple U-line balancing problem with travel time is denoted as the SULB-T problem. The SULB-T problem is modeled as follows.

Minimize M

Subject to:

$$\bigcup_{m=1}^{M} ST_m = S \qquad (1)$$

$$ST_m \bigcap_{m \neq m'} ST_{m'} = 0 \qquad \text{for all } m \qquad (2)$$

$$\sum_{x \in ST_m} t_x + g \times DT_m \leq C \qquad \text{for all } m \qquad (3)$$

For each task $y \in S$

Either: If $(x,y) \in P$, $x \in ST_m$, $y \in ST_n$, then $m \leq n$, for all x; (4)

Or: If $(y,z) \in P$, $y \in ST_n$, $z \in ST_p$, then $p \leq n$, for all z.

Location constraints (5)

The objective is to minimize M, the number of stations. Constraint 1 requires that all tasks be assigned to a station. Tasks are assigned to only one station as a consequence of constraint 2. Constraint 3 ensures that the sum of the task completion times and the travel time in a station does not exceed the cycle time. Constraint 4 imposes the following

condition on the tasks that may be assigned to a station. Only those tasks whose predecessors or successors have been assigned to the same or an earlier station are eligible for assignment to a station. Whenever a crossover is considered the current locations of the most recently assigned forward and backward tasks are used to determine the path. The path is checked for feasibility with respect to location constraints (5). Infeasibility occurs when stations are arranged so that travel in one station interferes with work and travel in another station.

While dynamic programming and integer programming approaches can be used to find an optimal solution to the SULB problem, dynamic programming cannot be used for the SULB-T problem because of the travel time. For this reason and because branch and bound was effective for the SULB problem (Part I of the thesis), it is used here for the SULB-T problem.

2.4 A Branch and Bound Algorithm

The following branch and bound algorithm computes an optimal balance for the SULB-T problem.

Step 1: Initial bounds on the number of stations

1.1 An Upper Bound, UB^M

Solve the SULB-T problem using a heuristic, H, to obtain an upper bound, $UB^M=H(x|x\in S)$, on the number of stations. Set the current upper bound $U_{current}=UB^M$. This solution represents the current best solution. While a few heuristics are available for the SULB problem (Miltenburg and Wijngaard [1994]), a single pass heuristic with priority based on task time is used in this work because

it is simple, fast and effective (Hackman, Magazine and Wee [1989]). Tasks from a set of assignable tasks are assigned to stations in decreasing order of task time according to the following procedure. When a task is tentatively assigned to a station, its location on the U-line is determined and the co-ordinates of the starting point and ending point of the task are determined. The current travel distance is calculated. The travel time is determined and is added to the sum of the task completion times. If the result does not exceed the cycle time, the assignment becomes permanent, the station idle time is recalculated, and the set of assignable tasks is updated. The process is then repeated.

1.2 A Lower Bound, LB^M

Calculate a lower bound on the number of stations, LB^M.

$$LB^M = \lceil \{ \sum_{x \in S} t_x + g[\sum_{x \in S} \alpha_x \ell_x + W] \}/C \rceil ,$$

where $\lceil z \rceil = z$ is the smallest integer $\geq z$. The expression contains two terms. The first term is the sum of the task completion times. The second term is a lower bound on the total travel time for the U-line found by assuming a single operator completes all tasks from beginning to end and returns to the beginning of the line. Thus the lower bound is the time to travel the task distances, $g\sum_{x \in S} \alpha_x \ell_x$, plus g times the U-line width, W.

If $UB^M = LB^M$ the heuristic solution is optimal and the algorithm stops.

Step 2: Branch and Bound Loop

2.1 Enumerate successor nodes to the branching node

Nodes represent stations (except for the first node which is a starting node).

In this step all nodes that follow the current node are enumerated using a modification of the node enumeration procedure of Van Assche and Herroelen [1978].

2.2 *Bounding*

For each node, i, select the largest of four lower bounds on the number of stations, $LB_i = \max(LB1_i, LB2_i, LB3_i, LB4_i)$. The first lower bound is based on the sum of the processing times of the unassigned tasks, and the other three are based on the distribution of the completion times of the unassigned tasks ($LB3_i$ and $LB4_i$ were proposed by Johnson [1988b]).

$$LB1_i = D_i + \lceil \{ \sum_{x \in NV_i} t_x/C + g[\sum_{x \in NV_i} \alpha_x \ell_x + LB(\text{return})] \}/C \rceil ,$$

$$LB2_i = D_i + \lceil \#(x \in NV_i)/\lfloor C/t_{min} \rfloor \rceil ,$$

$$LB3_i = D_i + \#(x \in NV_i \mid t_x > C/2),$$

$$LB4_i = D_i + \#(x \in NV_i \mid t_x > 2C/3) + 0.5\#(x \in NV_i \mid C/3 < t_x \leq 2C/3),$$

where

$\lfloor x \rfloor$ = The integer portion of x,

D_i = The depth of node i—the number of nodes or stations in the path to and including node i,

NV_i = The set of tasks not yet assigned to stations,

$\#(x \in NV_i)$ = The number of tasks in NV_i,

t_{min} = The average of the two minimum task times in NV_i, and

LB(return) = A lower bound on the return distance that results if all unassigned tasks are assigned to a single station.

Consider, for example, $LB4_i$. Each task with processing time greater than

(2/3)C cannot be added to a station having a task with processing time greater than C/3, and of the tasks with $C/3 < t_x \leq 2C/3$ no more than two can be added to the same station.

Fathom any node, i, where $LB_i \geq U_{current}$. In this case, node i cannot improve on the current best solution. If a node completes the U-line balance and the solution improves on the current best solution then revise $U_{current}$.

2.3 Branching

Branch on the deepest node—the node i with the greatest D_i. In the case of a tie, select the node with the smallest *accumulated task idle time per station*, A_i, in the path to the node;

$$A_i = (D_i \times C - \sum_{x \in V_i} t_x)/D_i,$$

where V_i = The set of tasks assigned to stations in the path to and including node i.

(Note that A_i does not consider travel time. A variation of this branching rule could consider travel time, although this might reward excessive travel. Alternate branching procedures is an area of ongoing research.)

2.4 Stopping Rule

Stop when there is no node i with $LB_i < U_{current}$. All nodes are fathomed and no further improvement is possible. The optimal solution is the current solution and the U-line is balanced with $U_{current}$ stations.

3. The Two U-line Balancing Problem

3.1 Definitions and Notation

In facilities where a number of U-lines are operating, stations may contain tasks on more than one U-line (Figure 2). These are called ***multi-line*** stations. This reduces the number of stations and, consequently, the number of operators required to operate the facility compared to the case where the U-lines are balanced separately. The simplest multiple U-line balancing problem—the problem of balancing two U-lines simultaneously—is considered in this paper. This is called the ***two U-line balancing problem with travel time***, the ***2ULB-T*** problem. The location of the multi-line stations is restricted to the openings of the U-lines. This restriction reduces the number of possible multi-line stations that must be considered when determining an optimal balance. It is also reasonable to locate multi-line stations here as this is where material enters and leaves the U-lines. Since stations cannot cross each other, there can be no more than two multi-line stations for any pair of U-lines. Figure 4 shows that when there are two multi-line stations, each station includes tasks from one side of each U-line. When there is only one multi-line station, it can include tasks from both sides of each U-line. Notice that the maximum reduction in the total number of stations is one station. Define the following notation.

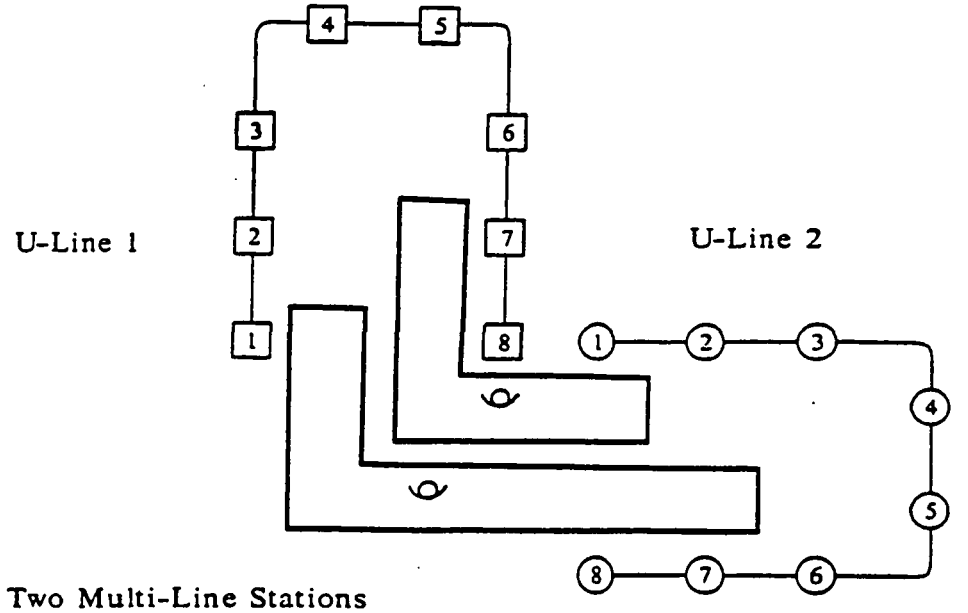
Variables specific to U-line u, u = 1, 2.

N_u = The number of tasks to be completed on U-line u.

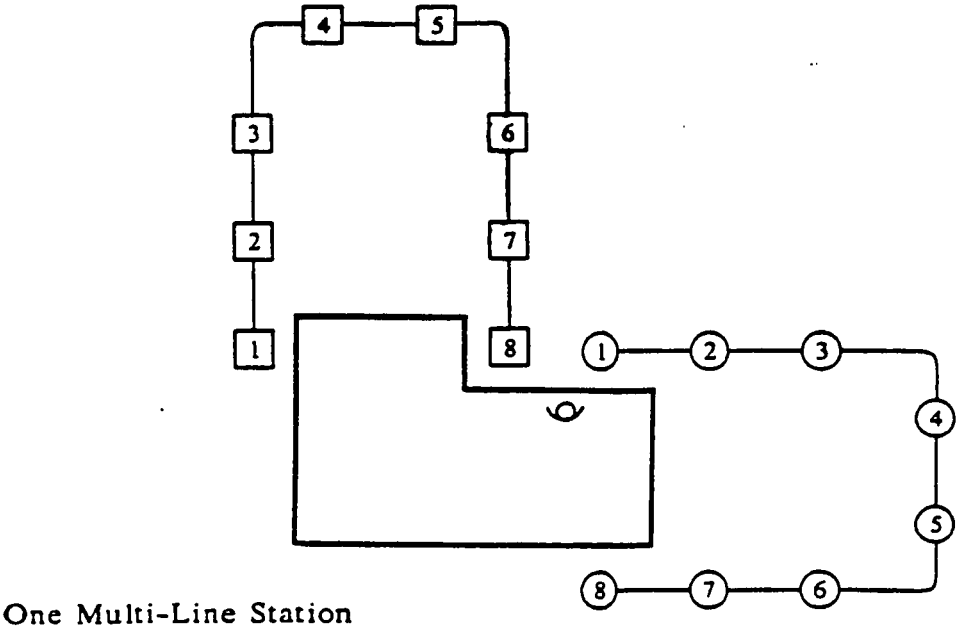
x(u) = Task x to be completed on U-line u. $x = 1, \ldots, N_u$.

Figure 4

Two U-Lines with Multi-Line Stations



Two Multi-Line Stations



One Multi-Line Station

S_u = {x(u) | x= 1,...,N_u}, the set of tasks on U-line u.

P_u = The set of precedence constraints for S_u.

$t_{x(u)}$ = The processing time for task x on U-line u.

$\ell_{x(u)}$ = The task distance on U-line u required by task x.

$\alpha_{x(u)}$ = The proportion of $\ell_{x(u)}$ to be included in travel time.

M^*_u = The minimum number of stations needed to balance U-line u.

$S\delta_{j(u)}$ = A subset j of tasks from U-line u which forms a feasible subset of tasks for assignment to one or two multi-line stations that removes δ single line stations from u, δ=1,2. These are called multi-line subsets.

$LB(\tau_{\delta(u)})$ = A lower bound on the sum of task times from U-line u which must be assigned to multi-line stations to reduce the number of single-line stations on u by δ.

L_u = The length of each of the front and back of U-line u.

W_u = The width of U-line u.

We assume that U-lines are constructed of straight lines and right angles and

$\sum_{x(u)\in Su} \ell_{x(u)} = 2L_u + W_u$ for u=1,2.

Variables specific to a station:

Let m(A) denote a station, A=1,2,12. m(1) and m(2) denote stations on a single line, A=1,2. m(12) denotes a multi-line station spanning U-lines 1 and 2, A=12.

M_A = The number of stations of type A.

$ST_{m(A)}$ = {x(u) | task x(u) is performed at station m(A)}.

$DT_{m(A)}$ = Travel distance in station m(A).

Variables specific to a problem instance:

C = The cycle time.

d = The minimum distance between the two U-lines at the multi-line station.

M_{total} = The total number of stations required to complete all tasks on both U-lines.

$$M_{total} = M_1 + M_2 + M_{12}$$

3.2 A Model for the 2ULB-T Problem

Minimize M_{total}

Subject to:

$$\left[\bigcup_{m=1}^{M_{12}} ST_{m(12)}\right] \cup \left[\bigcup_{m=1}^{M_1} ST_{m(1)}\right] \cup \left[\bigcup_{m=1}^{M_2} ST_{m(2)}\right] = S_1 \cup S_2 \qquad (6)$$

$$ST_{m(A)} \cap \underset{m(A)\neq m(A)'}{} ST_{m(A)'} = \varnothing, \qquad \text{all } m(A) \qquad (7)$$

$$\sum_{x(u)\in ST_{m(A)}} t_{x(u)} + g \times DT_{m(A)} \leq C, \qquad \text{all } m(A) \qquad (8)$$

$$M_{12} \leq 2 \qquad (9)$$

The assignment of tasks to single-line stations, m(1) and m(2), and multi-line stations, m(12), must satisfy the precedence constraints for each U-line. (10)

Location constraints (11)

The objective is to minimize the total number of stations needed to complete the tasks on both U-lines. All tasks on each U-line are either assigned to a station on their respective U-line or to a multi-line station through the first constraint. Constraint 7 prevents any task from being assigned to more than one station. Constraint 8 ensures that the sum of the task completion times plus the travel time in any station does not exceed the cycle time. No more than two multi-line stations are permitted by constraint 9. Constraint

10 ensures that the following precedence constraints are satisfied for each U-line.

- When there is one multi-line station, each task in the station has all predecessors or successors in the station.
- When there are two multi-line stations, one station contains tasks on the front (or back) of the U-line, each of which has all predecessors (or successors) in the station. The other station includes at least one task on the back (or front) of the U-line and every task in the station has all predecessors or all successors in the two multi-line stations.
- For tasks in a single line station all predecessors or successors are in the same station, an earlier station, or a multi-line station. (See Figure 4.)

Constraint 11 prevents infeasible crossovers.

3.3 An Optimal Solution Algorithm For The 2ULB-T Problem

Before presenting a detailed description of the optimal solution algorithm for the 2ULB-T problem, we give a general description of the four steps that comprise it.

Step 1: Balance the two U-lines separately to find the total number of stations required if no multi-line stations are allowed. Call the result the *balance for separate lines.*

Step 2: Check whether it is theoretically possible to improve the *balance for separate lines* by replacing two single-line stations, one from each U-line, with one multi-line station. If it is, use a branch and bound procedure to identify such a single multi-line station (if one exists). Call the result the *balance with one multi-line station.*

Step 3: Check whether it is theoretically possible to improve the *balance for separate lines* by replacing three single-line stations with two multi-line stations. If it is, use a

branch and bound procedure to identify the two multi-line stations and call the result the *balance with two multi-line stations.*

Step 4: Select the balance that has the minimum number of stations from those that have been determined—the balance for separate lines (step 1), the balance with one multi-line station (step 2), and the balance with two multi-line stations (step 3). In the case of a tie, select the balance that best satisfies some secondary criterion such as uniform idle time across the stations, smallest total travel time, and so on.

A more detailed description of the algorithm follows.

Step 1. Balance the two U-lines separately

Find M_1^* and M_2^*, the optimal number of stations on each line when the lines are balanced separately, using the branch and bound algorithm in section 2.4.

Step 2. A balance with one multi-line station

2.1 Compare $M_1^* + M_2^*$ to a lower bound on M(with one multi-line station)

A lower bound on the total number of stations when one multi-line station is used is,

$$LB^M = \lceil \{ \sum_{u=1}^{2} \sum_{x(u)=1}^{N_u} t_{x(u)} + g \times LB^{DT} \} / C \rceil, \text{ where}$$

$LB^{DT} = LB^{DT(1)} + LB^{DT(2)} + 2d$, is a lower bound on the total travel distance. LB^{DT} is comprised of lower bounds on the travel distance in each U-line and the minimum travel distance back and forth between the U-lines.

If $LB^M = M_1^* + M_2^*$ go to step 3. Otherwise continue.

2.2 For each U-line generate all subsets of tasks that can be assigned

to the one multi-line station

All feasible subsets of tasks that can be assigned to the one multi-line station from each U-line, u=1,2, are generated. They are denoted $S1_{j(u)}$, j=1,2,3,···. A modified version of the node enumeration procedure of Van Assche and Herreolen [1978] is used to enumerate these subsets. A feasible subset satisfies three conditions.

1. Each task in the subset must have all predecessors or successors already in $S1_{j(u)}$. (See Figure 4.)
2. The subset contains at least one task (or the result will be a balance for separate lines).
3. It must be possible to assign the remaining tasks, S_u-$S1_{j(u)}$, to M_u^*-1 single-line stations on line u.

A lower bound on the sum of task times for the tasks assigned to the one multi-line station from U-line u, $LB(\tau_{1(u)})$, follows from conditions 2 and 3.

$$LB(\tau_{1(u)}) = \text{Max}\{ \sum_{x(u)=1}^{N_u} t_{x(u)} + g \times LB^{DT(u)} - (M_u^*-1)C, \min_{x(u)\in S_u'} \{t_{x(u)}\} \}, \quad u=1,2,$$

where S_u' is the set of tasks on line u that have no predecessors or successors.

If $LB(\tau_{1(1)}) + LB(\tau_{1(2)}) + g \times 2d > C$, then no multi-line station exists that improves the balance for separate lines, and the algorithm proceeds to step 3 where two multi-line stations are considered.

Upper bounds on the sum of the task times for the tasks assigned to the multi-line station from each U-line are:

$$UB(\tau_{1(1)}) = C - LB(\tau_{1(2)}) - g \times LB^{DT(m(12))}, \quad \text{and}$$

$$UB(\tau_{1(2)}) = C - LB(\tau_{1(1)}) - g \times LB^{DT(m(12))}, \quad \text{where}$$

$LB^{DT(m(12))}$ is a lower bound on the total travel distance in the multi-line station, m(12).

$$LB^{DT(m(12))} = \ell_{x(1)'}(\alpha_{x(1)'}+1) + \ell_{x(2)'}(\alpha_{x(2)'}+1) + 2d.$$

$x(u)' \in S_u'$ is the task with the $\min_{x(u)\in S_u'}\{\alpha_{x(u)}\ell_{x(u)}\}$ from each U-line, u = 1,2. d is the distance between the two U-lines. Each subset, $S1_{j(u)}$, u=1,2, j=1,2,3,··· must satisfy, $LB(\tau_{1(u)}) \le \sum_{x(u)\in S1_{j(u)}} t_{x(u)} \le UB(\tau_{1(u)})$

Note that two lists of subsets, one for each U-line, will be generated.

2.3 Find the optimal multi-line and single-line stations

The procedure for finding the optimal multi-line and single-line stations works as follows. The U-line u, with the minimum number of subsets available to be assigned to a multi-line station is selected and a subset $S1_{j(u)}$ is chosen from the list of multi-line subsets. The SULB-T problem with tasks S_u-$S1_{j(u)}$ is solved using the algorithm described in section 2.4 to find $M_u(S1_{j(u)})$, the number of stations on U-line u when the tasks in $S1_{j(u)}$ are assigned to the multi-line station. If $M_u(S1_{j(u)}) > M_u^*-1$, the subset is fathomed. All subsets of $S1_{j(u)}$ are also fathomed since they cannot produce balances with M_u^*-1 stations. If $M_u(S1_{j(u)}) = M_u^*-1$, then $S1_{j(u)}$ is denoted $S1_{j(u)^*}$ and the procedure searches for a multi-line subset from the other U-line, say line v, which gives $M_v(S1_{j(v)}) = M_v^*-1$ stations and requires

$$\sum_{x(v)\in S1_{j(v)}} t_{x(v)} \le C - \sum_{x(u)\in S1_{j(u)^*}} t_{x(u)} - g \times LB^{DT(m(12))}$$

units of time in the multi-line station. If no such subset is found, subset $S1_{j(u)^*}$ is fathomed and the process is repeated. To minimize the solution space evaluated, any time the subset search repeats it selects the U-line, u, with the minimum number of unfathomed subsets. Using binary search techniques when selecting the next subset from U-line u can reduce the number of subsets that are considered and thus the number of single U-line problems, S_u- $S1_{j(u)}$, which must be solved.

Step 3. A balance with two multi-line stations

3.1 Compare $M_1^* + M_2^*$ to a lower bound on M(with two multi-line stations)

Two multi-line stations are arranged in such a way that operators working in stations do not cross paths. For example, the multi-line stations shown in Figure 4 are arranged so that one station consists of tasks from the beginning of U-line 1 and the end of U-line 2. The other multi-line station has tasks from the end of U-line 1 and the beginning of U-line 2. Other arrangements would permit crossovers to occur. A lower bound on the number of stations in a balance with two multi-line stations is:

$$LB^M = \lceil \{ \sum_{u=1}^{2} \sum_{x(u)=1}^{N_u} t_{x(u)} + g \times LB^{DT} \} / C \rceil, \text{ where}$$

$LB^{DT} = LB^{DT(1)} + LB^{DT(2)} + 2(d_1+d_2)$, is a lower bound on the total travel distance. d_1 and d_2 are the distances separating the two ends of U-line 1 from the corresponding ends of U-line 2.

If $LB^M = M_1^* + M_2^*$ go to step 4. Otherwise continue.

3.2 For each U-line enumerate all subsets of tasks that can be assigned to either multi-line station

Two multi-line stations are to replace three single-line stations. The maximum number of single-line stations in the U-lines is $M^*_1 + M^*_2 - 3$ with one station removed from one U-line and two from the other. (The numbering of the U-lines is arbitrary.) $LB(\tau_{\delta(u)})$ denotes a lower bound on the time that must be removed from U-line u to reduce the required number of single-line stations by $\delta=1,2$.

$$LB(\tau_{\delta(u)}) = \sum_{x(u)=1}^{N_u} t_{x(u)} + [(M^*_u-\delta)/M^*_u]g\times LB^{DT(u)} - (M^*_u-\delta)C,\ \delta=1,2,\ u=1,2.$$

The bound assumes that the total travel time on line u is proportional to the number of stations.

For each U-line two sets of multi-line subsets are enumerated. Each set contains subsets $S\delta_{j(u)}$, $\delta=1,2$, $u=1,2$, $j=1,2,3,\cdots$, which will theoretically allow the remaining problem $S_u-S\delta_{j(u)}$ to be solved in $M^*-\delta$ stations. Each subset $S\delta_{j(u)}$ will be divided between the two multi-line stations and so must contain at least one task on both the front and back of the U-line. As in step 2.2 a modified version of the node enumeration procedure of Van Assche and Herroelen [1978] is used to generate these subsets. Upper and lower bounds on the sum of the task times, $T_{\delta(u)}$, for the tasks in the subsets are computed.

$$LB(T_{\delta(u)}) = Max[\, LB(\tau_{\delta(u)}),\ \underset{x(u)\in S^f_u}{Min}\{ t_{x(u)} \} + \underset{x(u)\in S^b_u}{Min}\{ t_{x(u)} \}] ,\qquad u=1,2,\ \delta=1,2$$

$$UB(T_{\delta(u)}) = \phi\times C - LB(T_{\phi(v)}) - g\times(LB^{DT(m(12))} + LB^{DT(m(12)')}),$$

where $\phi = 3-\delta$ and S^f_u and S^b_u are the sets of available tasks from the front and back of U-line u that can be assigned to multi-line stations m(12) and m(12)′.

3.3 Find the optimal multi-line and single-line stations

A U-line, say u, is arbitrarily selected. A search is undertaken for a subset $S2_{j(u)}$ which allows two single-line stations to be eliminated from u and the SULB-T problem with tasks S_u-$S2_{j(u)}$ to be balanced with M^*_u-2 stations. If the search is successful the procedure turns to the other U-line, v, and searches for a subset, $S1_{k(v)}$ such that the tasks in $S2_{j(u)}$. and $S1_{k(v)}$ may be divided between two feasible multi-line stations. Each subset must be divided into a set of tasks with at least the first task at the front of the U-line, $S\delta^f_{j(u)}$, and another beginning at the back of the U-line, $S\delta^b_{j(u)}$. If, for example, we assume that the multi-line stations join the tasks at the front of one line to the back of the other then

$$\sum_{x(v)\in S2^f_{j(u)}} t_{x(v)} + \sum_{x(u)\in S1^b_{k(v)}} t_{x(u)} + g(DT^{m(12)}) \le C \text{ and } \sum_{x(v)\in S2^b_{j(u)}} t_{x(v)} + \sum_{x(u)\in S1^f_{k(v)}} t_{x(u)} + g \times DT^{m(12)'} \le C,$$

where m(12) is the first multi-line station and m(12)′ is the second. The subset $S1_{k(v)}$ must also permit $\delta = 1$ single-line station to be eliminated from line v; that is, the tasks in S_v-$S1_{k(v)}$ can be assigned to M^*_v-1 single-line stations. If no $S1_{k(v)}$ achieves M^*_v-1 then $S2_{j(u)}$ is fathomed along with any $S2_{g(u)} \subset S2_{j(u)}$ and the process is repeated.

Step 4. Select the best balance

Select the balance from the balance for separate lines (step 1), the balance with one multi-line station (step 2), or the balance with two multi-line stations (step 3), that has the minimum total number of stations. This is the optimal balance. In the case of a tie, select the balance that best satisfies some other criterion such as

uniform idle time across the stations, smallest total travel time, and so on. The algorithm is illustrated with an example in the appendix.

3.4 Computational Complexity

The simple U-line balancing, SULB, problem is NP hard. Consequently, the SULB-T problem and the 2ULB-T problem are also NP hard. Consider an instance of the 2ULB-T problem. The computational effort required to enumerate all feasible subsets in the solution of a simple U-line, line u, is $O(N_u^2[\#(S_u^f)]^2)$ where $\#(S_u^f)$ = the number of forward feasible subsets (Part I of this thesis). In solving the SULB-T problem N_u and $\#(S_u^f)$ are variables that depend on 1) the U-line itself—number of tasks, task completion times, width and density of the precedence graph, etc.—and 2) the subset of tasks removed from line u and assigned to the multi-line station, $S\delta_{j(u)}$, in that tasks S_u-$S\delta_{j(u)}$ remain to be assigned to stations on line u.

For any 2ULB-T problem many such simple U-line problems will be solved. The exact number depends on the characteristics of the problem instance. Suppose the optimal balance for a 2ULB-T problem instance requires one multi-line station. If there are n_1 and n_2 subsets of type $S1_{j(1)}$ and $S1_{j(2)}$ an upper bound of n_1n_2 may be placed on the number of SULB-T problems that have to be solved to identify the optimal balance. If the number of U-lines to be balanced simultaneously is increased to Z then the upper bound on the number of SULB-T problems which must be solved increases to $n_1n_2 \cdots n_Z$.

If the 2ULB-T problem is to be solved with two multi-line stations on U-lines with $n1_1$, $n2_1$, $n1_2$ and $n2_2$ subsets of type $S\delta_{j(u)}$,then the upper bound on the number of SULB-T

problems to be solved becomes $n2_1n1_2+n1_1n2_2$. For each SULB-T problem the additional problem of partitioning the subsets into two multi-line stations must also be solved. The curse of dimensionality makes it unlikely that any multiple U-line balancing problems beyond the 2ULB-T problem may be solved optimally.

4. Computational Results

An empirical study was undertaken for two reasons; to determine whether it is necessary to evaluate travel explicitly during line balancing and to gain insight into the problem of balancing two U-lines simultaneously.

The importance of considering travel distance

The first study was structured along the lines of a study done by Johnson [1982]. Sets of 50 randomly generated problems were used to evaluate methods of including travel in SULB algorithms. Since U-lines tend to be small (see Japan Management Association [pp. 122-125, 1987], Schonberger [p. 141, 1982] and Wantuck [pp. 132-162, 1989]) the problems sets contained 10, 12 or 15 tasks. As order strength is a significant factor in line balancing, problem sets were generated with order strengths .2, .5 and .8 ($\pm$.025) for each size. The characteristics of the problems solved may be found in Table 1.

In the empirical study, each problem was solved first by considering travel explicitly using the SULB-T algorithm of Section 2.4. Next travel was included implicitly and the SULB problem was solved using the branch and bound algorithm of section 2.4, with the travel provisions removed. Three methods of implicitly including travel in task time were considered:

1. Task time for each task was increased by a factor β.
2. Task time for each task was increased by a multiple of the average task distance on the U-line.
3. Task time for each task was increased by a multiple of the individual task distance.

The resulting task times used in the SULB problem are shown in Table 2.

For each of these methods six β values were tested. Every problem in the set was solved with two different cycle times. For all problems α_x was assumed to be 1. Once a solution was found to a problem, each station was analyzed and actual station travel and total time were calculated. Station feasibility relative to cycle time and location constraints was checked. Some partial results are shown in Table 3 and Figures 5 and 6. Several conclusions may be drawn.

1. There is a computational cost associated with considering task time explicitly. For most sets the time required to solve the 100 SULB-T problems was at least twice that for the SULB problems.

2. The computational time reductions for implicit methods are derived at the expense of individual station feasibility and poorer quality solutions. There are two reasons for explicitly including travel in U-line balancing algorithms.

i) to avoid stations which exceed cycle time because of inaccurately estimated travel time, termed cycle time infeasibility.

ii) to avoid infeasible station travel paths, termed location infeasibility. (In this study any crossover station which extended beyond the midpoint of the line was

Table 1

Part 1: Problem Set Characteristics

Characteristic	Value
Number of tasks	10, 12, 15
Order Strength	0.2, 0.5, 0.8
Task times, t_x	u.d. (1-100)
Task length, ℓ_x	u.d (2-10)
U-line width	u.d. (10-15)

A set of 50 problems was created for each number of tasks/order strength combination.

Part 2: Study Parameters:

Parameter	Values tested
Cycle time	u.d.(120-200), u.d.(200-300)
Travel time per unit distance, g	0.15, 0.3, 0.6
Beta values	Range dependent on the problem
α_x	1

Table 2

Three Methods of Implicitly Including Travel in U-Line Balancing

Method of including travel in task time	Task time used in the SULB problem
1. Multiply task time by a factor $\beta > 1$	$\beta \times t_x$
2. Increase task time by a factor of the time needed to travel the average task distance for the problem, l_{avg}.	$t_x + \beta \times g \times \ell_{avg}$
3. Increase task time for task x by a factor of the time needed to travel the individual task distance for task x, l_x.	$t_x + \beta \times g \times \ell_x$

where t_x = time for task x

β = weighting factor

g = travel time/unit distance

ℓ_{avg} = average task length

ℓ_x = length of task x.

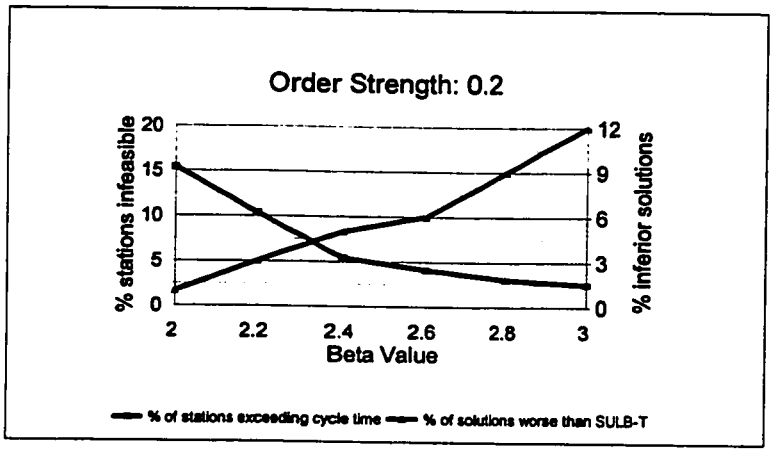
Table 3

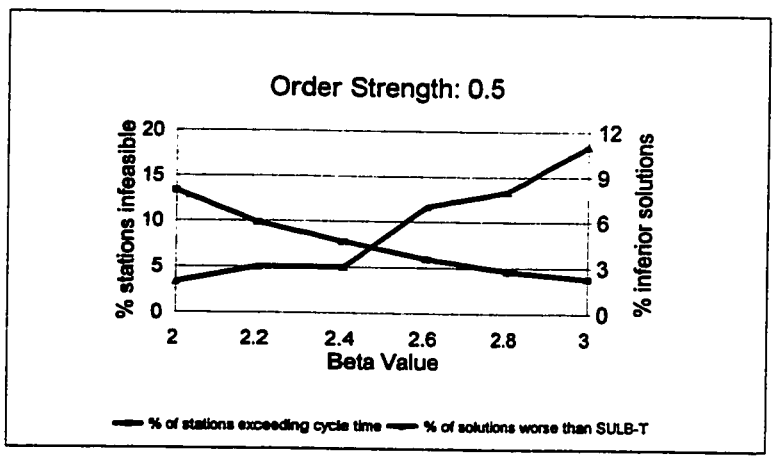
Results from 50 U-Line Balancing Problems each with Ten Tasks

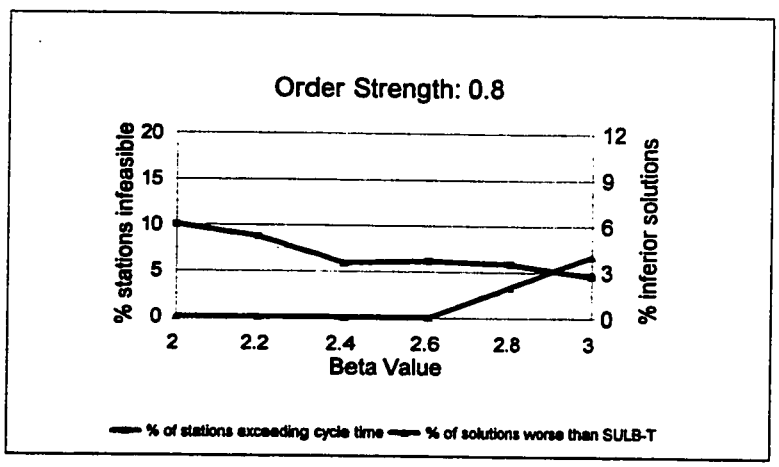
Travel Inclusion Method	Beta Value	Solution Time	Total # of Stations	# Cycle time Infeasible	# Location Infeasible	# Cycle time and Location Infeasible	m > msulbt (inferior solution)	m < msulbt (infeas. solution)
			ORDER STRENGTH 0.2					
SULB-T		718	383	0	0	0	0	0
SULB								
$\beta \times t_x$	1.05	355	375	116	7	4	1	9
	1.06	181	379	86	9	3	2	6
	1.07	175	383	60	5	0	3	3
$\beta \times g \times \ell_{avg}$	2.00	234	382	59	13	3	1	2
	2.20	348	385	40	9	1	3	1
	2.40	308	388	21	8	1	5	0
$\beta \times g \times \ell_x$	2.00	210	382	49	9	1	2	3
	2.20	300	386	32	9	0	4	1
	2.40	305	388	27	10	0	5	0
			ORDER STRENGTH 0.8					
SULB-T		94	393	0	0	0	0	0
SULB								
$\beta \times t_x$	1.05	40	382	90	28	9	0	11
	1.06	42	383	71	27	7	0	10
	1.07	36	386	50	22	3	2	9
$\beta \times g \times \ell_{avg}$	2.00	31	388	39	23	3	0	5
	2.20	30	388	34	25	2	0	5
	2.40	35	389	23	23	1	0	4
$\beta \times g \times \ell_x$	2.00	34	389	36	31	2	0	4
	2.20	35	390	29	32	3	0	3
	2.40	36	390	23	29	1	0	3

Figure 5

The Effect of Order Strength on Beta Selection



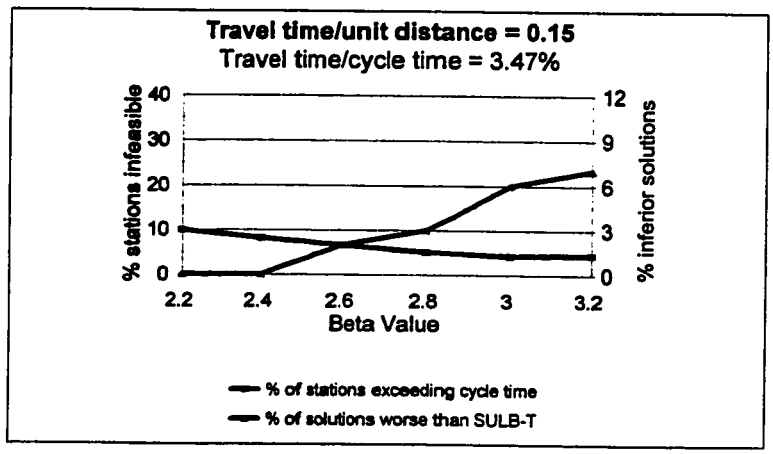




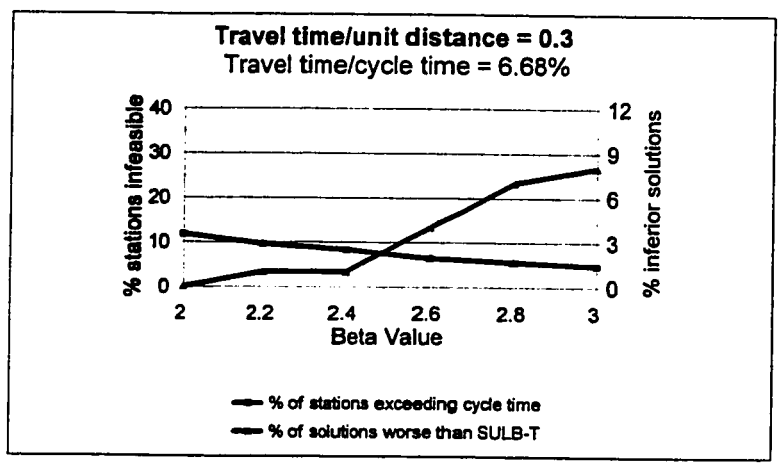
Number of tasks = 10
g = .3 sec/unit
Cycle time u.d.(120-200)
Travel included implicitly as a factor of average task length

Figure 6

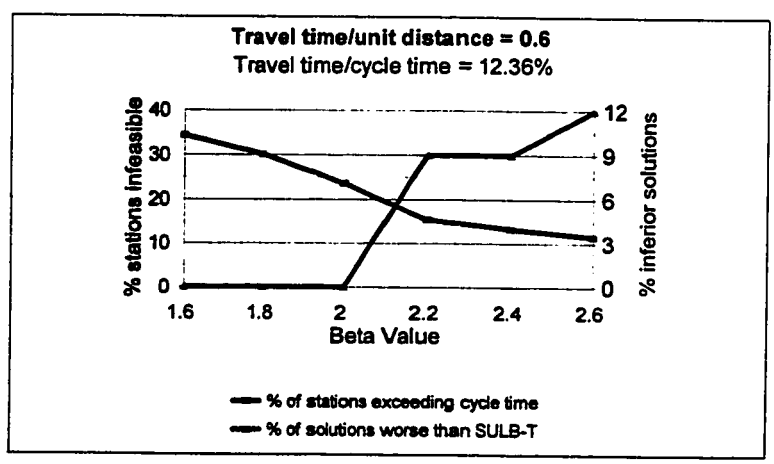
The Effect of Altering The Travel Time/Task Time Ratio



Optimal SULB-T solution = 469 stations



Optimal SULB-T solution for g = 0.3: 490



Optimal SULB-T solution for = 525 stations

Number of tasks = 12 Cycle time u.d.(120-200)

assumed to include infeasible travel paths)

Cycle time infeasibility was more common than location infeasibility.

3. The effectiveness of implicit methods depends on several factors:

   i. The value of β: There is a tradeoff between cycle time infeasibility and inferior solutions (those with more stations than in the SULB-T solution). Increasing the value of β reduces the number of stations infeasible relative to cycle time but increases the number of solutions containing more stations than the optimal SULB-T solution (see Figures 5 and 6). The issue is further complicated by the observation that the effectiveness of β values depends on the order strength of the problems solved. Note in Table 3 and Figures 5 and 6 that no single value of β is optimal for all problem sets. The optimal beta value varied directly with order strength and inversely with the ratio of travel time to cycle time. It was also found to vary inversely with the number of tasks in the problem. Adjusting β to fit different problem characteristics adds another dimension to implicitly including travel.

   ii. The ratio of travel time to cycle time: In the study this ratio was increased by raising the value of g, the travel time/unit distance. Figure 6 illustrates the effect of changing the ratio. Not only does the performance of the algorithm deteriorate significantly as the ratio increases but the optimal value for β differs depending on the value of g (or the value of the ratio). The travel time/cycle time ratio is affected by several factors:

      a) The travel time/unit distance, g,

b) The task lengths, ℓ_x,

c) The proportion of task length included in travel time, α_x and

d) The task time/travel time ratio.

A change in any one of the factors affects ratio of travel time/cycle time altering the performance of the algorithm.

4. The approaches which consider task distances outperformed the approach of increasing task time by some factor. A less intuitive result is that increasing task times by a factor of average task distance performed as well or better than using a factor of individual task distances.

5. In individual stations the implicitly calculated station times were relatively close to actual station time. For example for a problem with 10 tasks, g = 0.3, order strength = 0.2 and β = 2.4 the average absolute difference between the two was less than 2% with 97% less than 5%. However, individual stations varied widely with deviations ranging from -7.4% to 18%.

6. Location feasibility also contributed significantly to station infeasibility. The constraint used in this study, that crossover stations cannot occur once either forward or backward tasks extend beyond the midpoint of the line is relatively loose. Stronger location constraints will only provide greater incentive for using the SULB-T algorithm proposed.

2ULB-T problems of practical size

The second study was undertaken to examine the problem of balancing two U-lines simultaneously. For this study a mixture of U-lines ranging in size from 7 to 18

tasks was selected as the problem set. Five were from the SALB literature (Bowman) [1960], Dar-El [1964], Jackson [1956], Jaeschke [1964] and Merten [1967])[1] and 23 were randomly generated. The parameters and characteristics of these test problems are described in Table 4. 20 pairs of problems were randomly selected from the 28 problems in the problem set. Each pair comprised a two U-line problem instance. Five different randomly generated cycle times, ranging from $C=\max\{t_{x(u)}+1\}$ to $C=2\max\{t_{x(u)}\}$, were selected for each of the 20 multiple U-line problems to create a total of $5\times20=100$ 2ULB-T problem instances. The algorithm presented in sections 3.3 was used to determine the optimal balances. To simplify programming only balances with one multi-line station were considered for the 2ULB-T problem. (This is equivalent to the two U-line balancing approach taken at Toyota, Cambridge where a single project leader is responsible for work shared between two U-lines while team leaders and members complete work on their respective lines). Two performance measures were tracked—the number of stations in the optimal balance, and the computational effort required to obtain the optimal balance. The following results were obtained.

1. 30 of the 100 2ULB-T problems had one multi-line station in the optimal balance. 70 did not. That is, the optimal balance to the 70 2ULB-T problem instances was comprised of the optimal balances to two separate SULB-T problem instances. 32 of the 70 were already at the lower bound for the

---

[1] The authors thank Professor James Patterson of Indiana University for providing us with these five problems.

Table 4

Problems In The Second Study

Parameter	Random Problems
Number of problems	23
Number of tasks, N	u.d.(7-18)[1]
Maximum task time, t_{max}	u.d.(10-35)
Task processing time, t_x	u.d.(1 to t_{max})
Task distances, ℓ_x	u.d.(2-10)
U-line width, W	u.d.(10-15)
length, L	$L = .5(\sum_{x \in S} \ell_x - W)$
Distance between U-lines, d	u.d.(4-20)
Maximum number of successors, s_{max}	u.d.(3-7)
α_x=.8 for all tasks	
g=.3	
Rule for eliminating redundant paths	Eliminate later relation

1. A uniform distribution over an interval from 7 to 18 tasks.

Characteristic[2]	Average, Minimum, Maximum
Number of Tasks	11.6, 7, 18
Number of Precedence Relations	14.6, 6, 28
Width of Precedence Graph	3.4, 2, 6
Order Strength of Precedence Graph	.45, .66, .89

2. Characteristics for five problems from the literature and 23 randomly generated problems.

number of stations. The average reduction in the number of stations in the optimal balance as a result of the multi-line stations was 4 percent for all 100 instances. The average reduction for the 30 instances where the optimal balance included a multi-line station is 11.5 percent.

2. For small U-lines, like the ones considered in this study, the computational effort required to obtain the optimal balance for the 2ULB-T problems was moderate. All problem instances were solved in less than 5 minutes. The average upper bound on the number of SULB-T problem instances that needed to be solved in a 2ULB-T problem instance was 17.3 problems. (This bound is easily determined once all the feasible subsets of tasks, $S\delta_{j(u)}$, that can be assigned to the multi-line station are enumerated.) The upper bound was quite loose in this study. The average number of instances that were actually solved was 2.75. This was due in part to the use of binary search techniques in steps 2.3 and 3.3 of the algorithm.
3. While the effectiveness of heuristics for SULB-T and 2ULB-T problems was not investigated in this study, it is worth noting that the maximum task time heuristic (used in the solution algorithms for both problems to obtain an initial upper bound on the number of stations) produced excellent results. Tightening the lower bound on the number of stations will improve the recognition of instances when the heuristic has actually achieved the optimal. The development of heuristics for more complex multiple U-line problems is a promising area for future research.

5. Discussion and Conclusions

The empirical studies illustrate that balancing two U-lines together can increase operator efficiency. They also provide a rationale for including travel during U-line balancing. The additional computational effort is a small price to pay for the increase in accuracy and reduction in line balance adjustments which would be required using implicit methods. A number of other considerations may provide a further incentive to explicitly consider travel in U-line balancing.

Restrictions on the location of particular tasks

In many manufacturing settings, the location of a task is fixed or limited to a specific section of a production line because of machine and material handling constraints. The algorithms in this paper can be modified to include constraints on the locations for these tasks. Any assignment of tasks to stations that violates these constraints is infeasible. To illustrate, consider again the U-line in Figure 3. The line is 73 feet long. Let $0 \le u \le 73$ be a position along the line; with $u=0$ being the beginning of the line and $u=73$ the end. Suppose that tasks 1 and 3 are predecessors of task 4, and tasks 8, 9, 10, 12 and 14 are successors. $\ell_1+\ell_3 = 13$ and $\ell_8+\ell_9+\ell_{10}+\ell_{12}+\ell_{14} = 32$. Define B_4 to be the beginning location of task 4. Then

$$13 \le B_4 \le 73-32=41.$$

Now suppose that task 4 is to be performed on a machine that must be located between $u=10$ and $u=22$. The machine is $\ell_4=5$ feet long. Then $B_4 \ge 10$ and $B_4+\ell_4 \le 22$, or $B_4 \le 17$. The resulting constraint on the beginning location of task 4 is: $13 \le B_4 \le 17$.

If task 1 is assigned first, it occupies the first five feet of the U-line. If task 2 is assigned next, then a lower bound on B_4 is $(\ell_1+\ell_2)+\ell_3 = 24$, as task 3 must still be completed prior to starting task 4. This violates the location constraint and so task 2 cannot be assigned. It is quite straight-forward to add this logic to the branch and bound algorithms in this paper.

U-lines with different cycle times

Consider a 2ULB-T problem where two the U-lines, 1 and 2, have different cycle times, $C_1 \neq C_2$. Assume that $C_1 < C_2$, and consider the following two possibilities.

1. C_2/C_1 is an integer. This could occur when one line produces parts for the other line, or both produce parts for the same downstream operation but the quantity of each part used is different. The problem may be solved as described in section 3 using C_1 as the cycle time for the multi-line station. The implementation of the balance will require the operator to travel to U-line 2 once every C_2/C_1 cycles to complete the tasks on U-line 2. The rest of the time the operator remains in U-line 1.
2. C_2/C_1 is not an integer. U-line 2 is visited every C_2/C_1 cycles on average. On those occasions when the time between visits exceeds C_2/C_1 cycles, some inventory will be needed in U-line 2 to prevent a shortage. The inventory will be replenished during the other occasions when the time between visits is less than C_2/C_1 cycles. Greater attention will have to be paid to scheduling trips to

U-line 2 to avoid shortages and fluctuations in inventory levels.

Travel time functions

In this paper travel time is a scalar multiple of travel distance. Other functions could be used to account for factors such as starting and stopping, changing lines, fatigue when travel distance exceeds an acceptable level, and so on.

Rotating operators through stations

Part of the Shojinka concept calls for the rotation of operators through different stations. The exact rotation depends on the skills of the operators, the training objective, operator preferences, the physical health of the operators, and so on. Travel distance may also be a factor. For example, the rotation may try to spread each operator's travel evenly over an 8-hour shift.

5.1 Extensions

U-shaped production lines are an important part of modern production systems. This paper extends the literature on these production lines in the two important ways. First, the simple U-line balancing problem is extended to include consideration of travel by operators in each station and the effect of location on station feasibility. Second, the multiple U-line balancing problem, wherein stations can include tasks from more than one U-line, is introduced through the 2ULB-T problem. The SULB problem is known to be NP hard. Consequently, the SULB-T problem and the 2ULB-T problem, which are more difficult problems, are also NP hard. The storage and

computational requirements for the branch and bound algorithms for these problems are high. More research into the computational requirements of optimal solution algorithms for these and other U-line problems is required. The development of heuristics appears to be a promising area for future research. There are many other areas where more research work can be done. Different branching procedures and bounds for the branch and bound algorithm can be investigated. Stochastic task completion times can be considered.

Designing facilities

It is not always the case that the front and back of a U-line are of equal length. The lengths of the front, back and side of a U-line and its shape are design parameters. During the process of designing a facility, different U-lines geometries may be analyzed. Each configuration gives a different set of SULB-T problems which can be solved for a range of possible cycle times. In this way the effect of different designs on the number of stations and operators, travel distances, required floor space, and so on, can be determined.

Other assembly line balancing problems

Shtub and Dar-El [1989] present a classification scheme for assembly systems in which assembly systems are organized along three dimensions: division of labour—individual assembly, workstations, or collective assembly; production flow—paced or unpaced; and product mix—single model line, mixed model line, or multi-model line. The traditional and U-line balancing problems are classified as

workstation, paced and single model assembly line problems. The difference between the two is the type of workstation employed—straight in the case of the traditional problem, and straight, crossover and multi-line in the case of the U-line problem.

The results in this paper can be extended to other assembly line problems and to non-assembly line problems. For example, mixed model and multi-model U-line problems, unpaced U-line problems, and collective assembly U-line problems can be investigated. Travel time and location constraints can be applied to special assembly line problems. Agnetis et al. [1995], for example, consider the problem of balancing segmented assembly lines with incompatibilities among operations and groupings of operations which must be performed together. They propose breaking a single line into segments and allowing stations to include tasks on more than one segment to improve productivity in a facility producing heaters for automobiles. This is not unlike separating a U-line into a front and back and creating crossover stations that consider travel time and satisfy location constraints.

While beyond the scope of this paper, it is appropriate to note that there is a relationship between the multiple U-line problem and cellular manufacturing (also called group technology). Hall [1983] and Wantuck [1989] both use the terms cellular manufacturing and group technology when they discuss multiple U-lines. Burbidge [p. 5, 1991] defines group technology as "a method of organization for factories in which organizational units known as "groups" each complete a particular set or "family" of parts with no backflow, or crossflow between groups ...". Identifying the groups (i.e. machines and parts) is an important problem in group technology. The mixed model or

multi-model multiple U-line problem would take the groups identified by group technology, organize the machines in each group into stations on a U-line, and link the U-lines with stations that cross neighbouring lines. See the special issue of the Journal of Operations Management [1991] on "Group Technology and Cellular Manufacturing" for a good review and recent list of references on the subject.

More complex multiple U-line balancing problems

The positions of multi-line stations are restricted to the opening of the U-lines. Relaxing this restriction so that multi-line stations may be placed at other parts of the U-lines gives a more difficult problem. This is an area where more research may be done. Another area of ongoing research is the extension of the 2ULB-T problem to the general case of simultaneously balancing N U-lines. Travel distances and location constraints are more complex in this problem. It is unlikely that optimal balances can be obtained for other than small problem instances and so heuristic solution algorithms need to be developed.

References

Agnetis A., A. Ciancimino, M. Lucertini and M. Pizzichella, "Balancing Flexible Lines for Car Component Assembly," *Int. J. Prod. Res.*, 33, 2, 333-350 (1995).

Bowman, E.H., "Assembly Line Balancing by Linear Programming", *Oper. Res.*, 8, 3, 385-389 (1960).

Burbidge, J.L., "Production Flow Analysis for Planning Group Technology", *Journal of Operations Management*, 10, 1, pp. 5-27 (1991).

Dar-El, E.M., "Assembly Line Balancing - An Improvement on the Ranked Positional Weight Technique," *Journ. of Indust. Engineering*, 15, 2, 73-77 (1964)

Hackman, S.T., M.J. Magazine and T. S. Wee, "Fast, Effective Algorithms For Simple Assembly Line Balancing Problems," *Oper. Res.*, 37, 6, 916-924 (1989).

Jaeschke, G., " Eineallgemaine Methods Zur Losung Kombinatorisher Probleme," *Ablaufund Planungforschung*, 5, 133-153 (1964).

Jackson, J.R., "A Computing Procedure for a Line Balancing Problem", *Mgmt. Sci.*, 2, 3, 261-271 (1956).

Merten, P., "Assembly Line Balancing by Partial Enumeration", *Ablaufund Planungforschung*, 8, 429-433 (1967).

Miltenburg G.J. and J. Wijngaard, "The U-Line Balancing Problem." *Mgmt. Sci.*, 40, 10, 1378-1388 (1994).

Monden, Y., *Toyota Production System*, Second Edition, Industrial Engineering Press,

Institute of Industrial Engineers, Norcross, GA (1993).

Schonberger, R.J., *Japanese Manufacturing Techniques, Nine Hidden Lessons in Simplicity*, The Free Press, New York, 1982.

Shtub, A. and E.M. Dar-El, "A Methodology for the Selection of Assembly Systems", Int. J. Prod. Res., 27, 1, 175-186 (1989).

"Special Issue on Group Technology and Cellular Manufacturing", *Journal of Operations Management*, 10, 1, 1991.

Van Assche, F. and W.S. Herreolen, "An Optimal Procedure for the Single-Model Deterministic Assembly Line Balancing Problem," *Eur. J. of O. R.*, 3, 142-149 (1978).

Wantuck, K.A., *Just In Time for America*, The Forum Ltd., Milwaukee, WI, 1989.

Appendix

An Illustrative Example of the 2ULB-T Problem

A balance is required for the two U-lines described in Table 5. The lines are arranged as previously shown in Figure 4. The algorithm of section 3.3 is applied as follows to obtain the optimal balance.

Step 1. Balance the two U-lines separately

U-line u=1: $UB^M = H(x|x\in S) = 4$ stations. (H is the *maximum task time heuristic.*)

$$LB^M = \lceil \{ \sum_{x\in S} t_x + g(\sum_{x\in S} \alpha_x \ell_x + W)\}/C \rceil,$$

$$= \lceil \{ 219 + .3(.8\times 64 + 10)\}/ 70 \rceil$$

$$= 4 \text{ stations.}$$

Since $LB^M = UB^M$ the heuristic balance is optimal; $M_1^* = 4$.

U-line u=2: $UB^M = H(x|x\in S) = 4$ stations.

$$LB^M = \lceil \{179 + .3(.8\times 48 + 10) \} / 70 \rceil$$

$$= 3 \text{ stations.}$$

$LB^M < UB^M$ the branch and bound SULB-T algorithm searches for a solution that has 3 stations. None exists and so the heuristic solution is optimal; $M_2^* = 4$.

Step 2. A balance with one multi-line station

2.1 Compare $M_1^* + M_2^*$ to a lower bound on M(with one multi-line station)

$LB^{DT(u)} = W + \sum \alpha_{x(u)} \ell_{x(u)}$ for u=1,2. That is; $LB^{DT(1)} = 10 + .8\times 64 = 61.2$ feet, and

$LB^{DT(2)} = 10 + .8\times 48 = 48.4$ feet.

Table 5

Data For Illustrative Example

Product or U-line, u	U-line Geometry		Task			Immediate Predecessors
	L feet	W feet	x(u)	$t_{x(u)}$ seconds	$\ell_{x(u)}$ feet	
1	27	10	1	14	4	--
			2	34	4	--
			3	23	8	1
			4	15	6	2, 3
			5	21	5	3
			6	31	4	5
			7	9	6	4, 5
			8	20	6	6, 7
			9	8	10	8
			10	16	4	--
			11	28	7	9, 10
2	19	10	1	9	5	--
			2	27	3	1
			3	8	3	1
			4	38	7	1
			5	20	7	2, 4
			6	12	6	3, 4
			7	23	5	4
			8	24	8	2, 6, 7
			9	18	4	5, 7

C = 70 seconds
$\alpha_{x(u)}$=.8 for all x(u)
g=.3 seconds/foot
d=10 ft.

Then $LB^{DT} = LB^{DT(1)} + LB^{DT(2)} + 2d$. So;

$$LB^{DT} = 61.2 + 48.4 + 2\times10 = 129.6$$

A lower bound on the number of stations is

$$LB^{M} = \lceil \{ \sum_{u=1}^{2} \sum_{x(u)=1}^{N_u} t_{x(u)} + g\times LB^{DT} \} / C \rceil = \lceil \{398+.3\times129.6)\}/70 \rceil = 7.$$

Since $LB^{M} < M_1^{*} + M_2^{*}$ proceed to search for a multi-line station.

2.2 For each U-line generate all subsets of tasks that can be assigned to the multi-line station

• *Bounds*

The lower and upper bounds on the sum of the task times for the tasks assigned to the multi-line station from each U-line are:

$$LB(\tau_{1(u)}) = Max\{ \sum_{x=1}^{N_u} t_{x(u)} + g\times LB^{DT(u)} - (M_u^{*}-1)C, \ \min_{x(u)\in S_u'} \{t_{x(u)}\} \}, \ u=1,2,$$

$$\rightarrow LB(\tau_{1(1)}) = Max\{(219 + .3\times61.2 - 3\times70), \ 14\} = 27.4 \text{ sec.}$$

$$\rightarrow LB(\tau_{1(2)}) = Max[(179 + .3\times48.4 - 3\times70), \ 9\} = 9 \text{ sec.}$$

$$UB(\tau_{1(1)}) = C - LB(T_2) - g\times LB^{DT(m(12))}$$

$$\rightarrow UB(\tau_{1(1)}) = 70 - 9 - .3(4\times(.8+1) + 4\times(.8+1) + 2\times10) = 50.7$$

$$UB(\tau_{1(2)}) = C - LB(T_1) - g\times LB^{DT(m(12))}$$

$$\rightarrow UB(\tau_{1(2)}) = 70 - 27.4 - .3(4\times(.8+1) + 4\times(.8+1) + 2\times10) = 32.3$$

So $27.4 \le \sum_{x\in S_{1,j(1)}} t_{x(1)} \le 50.7$ and $9 \le \sum_{x\in S_{2,j(2)}} t_{x(2)} \le 32.3$.

The subsets on each U-line which fall within these bounds are:

$S1_{1,j}$: {1,2}, {1,3}, {1,9,11}, {1,11}, {2}, {2,10}, {9,11}, {10,11}, {11}

$S2_{1,j}$: {1}, {1,3}, {8}, {9}, {1,9}

As there are 14 subsets, there are at most 14 different SULB-T problems that need to be solved.

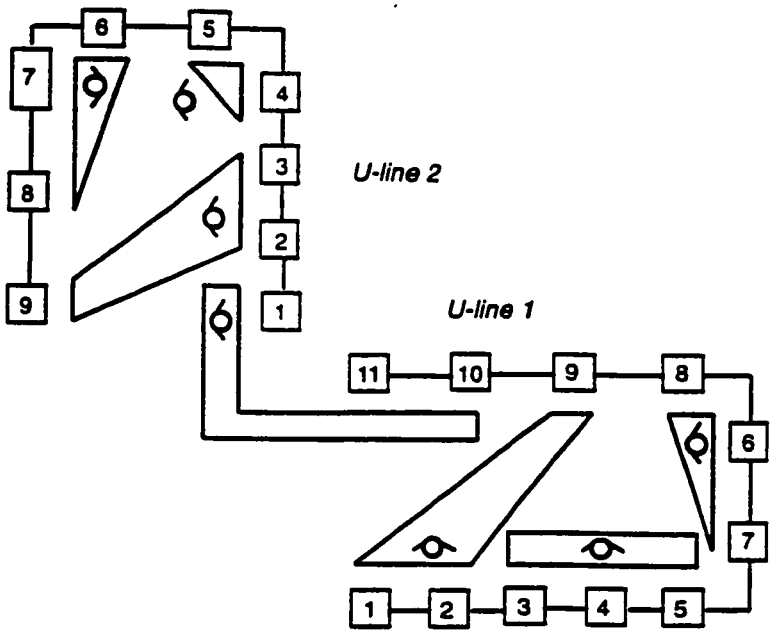
2.3 Find the optimal multi-line and single-line stations

Start with line u=2 because it has fewer subsets. Subset $S2_{1(1)}=\{1\}$ is selected. The remaining SULB-T problem has tasks S_2-{1}. Applying the algorithm of section 2.4 gives $M_2(S2_{1(1)}) = 3$, which is a reduction of one station from M_2^*. So the algorithm moves to the other U-line, u=1, and searches for a subset $S1_{1(1)}$ which gives $M_1(S1_{1(1)}) = M_1^*-1 = 3$ stations and whose total task time is less than $C - \sum_{x(u)\in S1\ j(1)} t_{x(u)} - g\times LB^{DT(m(12))} = 70 - 9 - 10.86 = 50.14$ seconds. The subset {10,11} is such a subset. The resulting optimal balance is shown in Figure 7.

Single-line stations:	U-line, u=1	Stations	1	2	3
		Tasks	{1,2,9}	{3,4,5}	{6,7}
		Travel time	13.7	10.3	7.4
	U-line, u=2	Stations	1	2	3
		Tasks	{2,3,9}	{6,7,8}	{4,5}
		Travel time	9.4	9.2	6.4
Multi-line station:		Station	1		
		Tasks	{10,11 from u=1, 1 from u=2}		
		Travel time	15.2		

Figure 7

Optimal Balance for Illustrative Example



BALANCING JUST-IN-TIME PRODUCTION UNITS: THE N U-LINE BALANCING PROBLEM

Abstract

U-shaped production lines are a common component of Just-In-Time systems. When a number of U-lines operate in close proximity an opportunity exists to balance two or more U-lines together and reduce the total number of stations required to operate the facility. The concept of a JIT production unit, where a number of U-lines produce and assemble parts for the same product line, is introduced. The problem of balancing a JIT Production Unit is considered as the N U-line balancing problem considering travel. The problem is modelled and heuristic solution algorithms are presented for both the general case, where U-line locations are not fixed, and for the fixed location case. A sub-problem, the problem of assigning tasks to stations which may include tasks from more than one U-line, provides new challenges for operations managers. This new problem is called the multi-line station assignment problem. The benefits of multiple U-line balancing are examined in an empirical study.

This paper was a co-winner of the CORS student paper competition in May 1995. The first revision has been re-submitted to INFOR for consideration for publication.

1. Introduction

In an increasingly competitive manufacturing environment many companies have implemented Just In Time (JIT) practices in their production systems to identify production inefficiencies and eliminate manufacturing waste (Monden [1993]). A major component of waste reduction efforts in JIT systems is *shojinka*, the process of continually striving to reduce waste in the workforce. Integral elements of shojinka are the use of U-shaped production lines, cross-training of operators and continual review and revision of work practices. The shape of U-lines improves visibility and allows the construction of stations containing tasks on both sides of the line. This arrangement, combined with cross-trained operators, provides greater flexibility in station construction than is available on a comparable straight production line. As a result, the number of operators needed on a U-line will be less than or equal to the number required on an equivalent straight line.

The problem of assigning tasks to stations on U-lines, the U-line balancing problem, was first modelled by Miltenburg and Wijngaard [1994]. The basic version is the Simple U-line Balancing (SULB) Problem which corresponds to the Simple Assembly Line Balancing (SALB) Problem on a straight assembly line. Because U-line stations may include tasks on both the front and back of the U-line a task may be assigned to a station if all of its predecessors or successors have been assigned to the same or an earlier station and there is sufficient idle time. Thus U-line balancing algorithms work through the task precedence relations for the problem in both directions simultaneously. The greater flexibility on U-lines comes at a price, U-line balancing problem is significantly more

complex than the straight line problem. For a SALB problem with N tasks and $\#(S^f)$ feasible subsets (created by working through the precedence relationship in one direction) the computational complexity is $O(N^2[\#(S^f)])$ (Queyranne [1985]). This compares to $O(N^2[\#(S^f)]^2)$ for a comparable SULB problem (Part I of this thesis). The increased complexity limits the size and difficulty of the U-line problems which may be solved optimally. Fortunately, since U-lines tend to be small, with 5-20 tasks in most instances, this limitation is not severely prohibitive for single problems.

The SULB problem has been solved using optimally using a recursive fixing dynamic programming (DP) algorithm (Miltenburg and Wijngaard [1994]), a reaching DP algorithm, and by depth-first and breadth-first branch and bound integer programming algorithms (Part I of this thesis). An effective breadth-first B&B algorithm for solving problems of typical size was identified in Part I.

Two other differences between U-lines and straight lines are essential to extending U-line balancing research, the time required for station travel and the necessity to avoid interference between operators moving in adjacent stations. While the total distance travelled by all workers on a straight line is fixed, regardless of the balance in effect, the total travel distance on a U-line is a variable dependent on the current station assignments. Since travel takes operator time, the sum of all task and travel times is also a variable dependent on the balance. The layout of the U-line and location of tasks restricts the ability of operators to cross from one side of the line to the other without interfering with nearby stations. To avoid such interference location constraints must be added to the problem formulation. In Part II of this thesis consideration of operator travel was

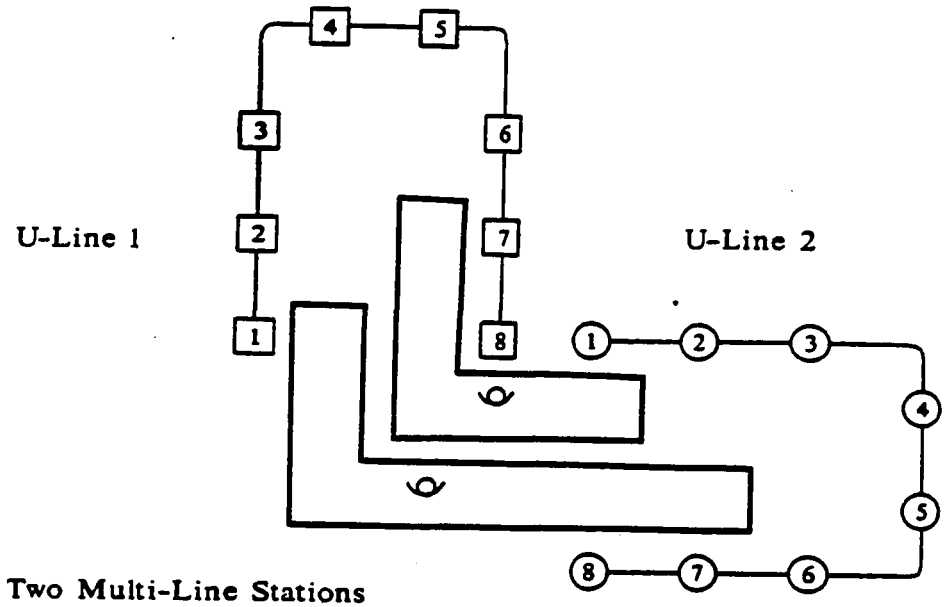
introduced into U-line balancing as the Simple U-line Balancing with Travel (SULB-T) Problem. Three distinct components to U-line station travel distance were recognized. Figure 1 in Part II shows examples of U-line balances and the travel distances associated with them.

Because U-lines are more flexible than straight lines the number of stations in a U-line balance will always be less than or equal to the number on a comparable straight line. If the U-line balancing problem is extended to allow the construction of stations which include tasks on more than one U-line further gains in productivity may be achieved. The advantages of multi-line stations are discussed in Schonberger [1982], Wantuck [1989] and Monden [1993]. Stations with tasks on two or more U-lines are called *multi-line stations*. In this problem, where operators may walk between U-lines during a cycle, a fourth travel distance, *multi-line travel*, is added. Part II introduced the problem of balancing two U-lines together where access to the U-lines is allowed only at the opening of the U. Because of this restriction, the number of multi-line stations in this Two U-line Balancing with Travel (2ULB-T) problem cannot exceed two without interference between stations (see Figure 1). If it is assumed that the distance between lines is no less than the width of the U-lines at most one station may be removed from the total number of stations required to balance the two U-lines. An optimal branch and bound algorithm was presented.

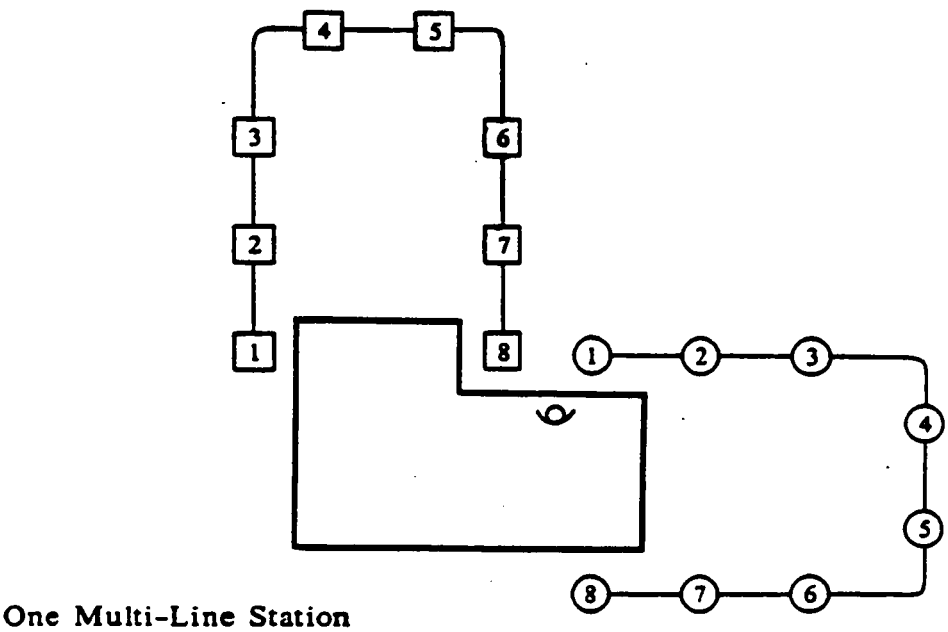
The ultimate objective of U-line balancing research would be to balance an entire production facility simultaneously, minimizing the total labour force. However, in a single JIT facility there may be several distinct types of products, each with its own production lines. Because the cycle time for any U-line is determined by demand for the

Figure 1

Two U-Lines with Multi-Line Stations



Two Multi-Line Stations



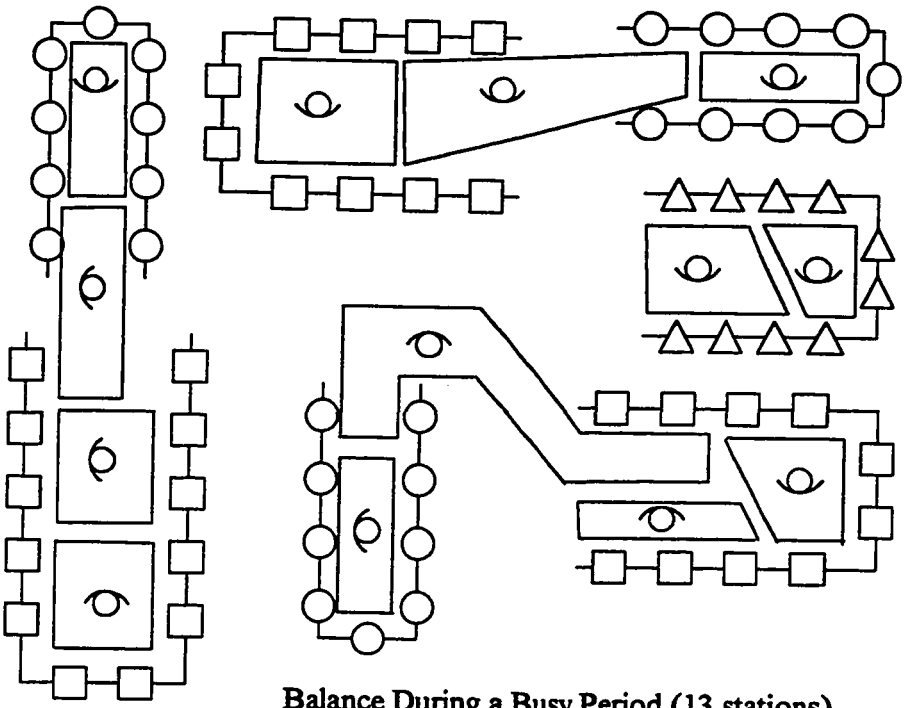
One Multi-Line Station

product produced on that line, U-lines producing different products will frequently operate with different cycle times. Those cycle times will change at different times corresponding to changes in demand. Therefore, it is not possible to balance an entire facility as a single unit. Instead, the concept of a *JIT Production Unit* is introduced. A JIT production unit is defined to be group of U-lines all producing parts for the same product line, operating at the same cycle time and located sufficiently close together that multi-line stations may be built connecting combinations of the U-lines. Figure 2 would be considered to be a JIT Production Unit. The U-lines all produce parts for the same product lines. As the production rate changes in response to demand, the lines must be re-balanced and the stations in the unit rearranged. Stations containing tasks on two or more U-lines are allowed in the solution and these stations provide the flexibility needed to minimize the number of stations required. The problem of balancing a JIT production unit with N U-lines while considering travel time will be called the N U-Line Balancing with Travel (NULB-T) Problem.

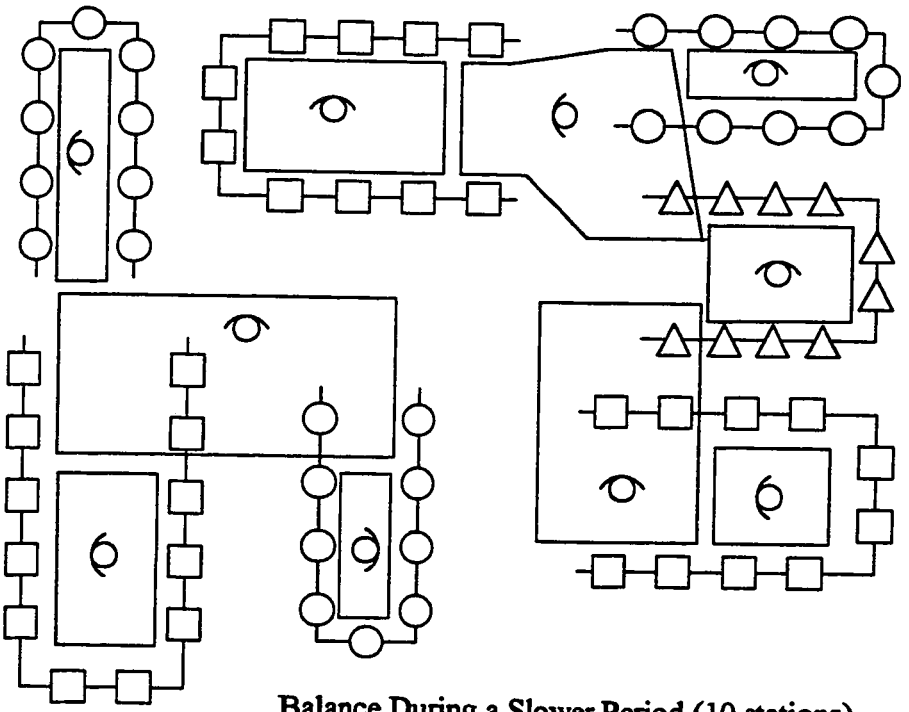
The SULB and SULB-T problems have been shown to be NP-hard and so the NULB-T problem which contains several SULB-T problems is also NP-hard. While it is possible to find an optimal solution for the two U-line balancing problem, the computational requirements are intensive. Determining an optimal solution for the NULB-T problem in a reasonable period of time is infeasible in all but the smallest problems. However, because of the flexibility in U-line station construction even simple heuristics have been shown to perform extremely well at finding optimal or near optimal solutions (Part I of this thesis). While it may be impossible to prove an optimal solution due to

Figure 2

An Example of a Just In Time Production Unit



Balance During a Busy Period (13 stations)



Balance During a Slower Period (10 stations)

computer memory and/or time limitations, the use of heuristic techniques should allow the identification of good solutions.

This paper extends previous work on U-line balancing by modelling the N U-Line Balancing with Travel (NULB-T) problem and presenting solution algorithms for it. Two variations of the problem will be considered. First, the general case, where there are no restrictions on the locations of the N U-lines, will be discussed. Second, the more restricted case in which U-line locations are fixed will be examined.

The remainder of the paper is organized in the following manner. In section 2 the model for the N U-line balancing problem is presented. In section 3, solution algorithms for the NULB-T problem are proposed. The procedure for solving both the general and fixed location NULB-T problems is identical for two of the three steps in the algorithm. This common procedure is contained in Section 3.1. In section 3.2, a procedure for multi-line station formation for the general case is presented. In section 3.3, the problem of assigning tasks to multi-line stations where U-line locations are fixed is considered and a solution algorithm is offered. Computational complexity of U-line balancing problems is compared in section 3.4. The performance of the NULB-T algorithm is evaluated in an empirical study is presented in section 4, followed by a discussion in the last section.

2. A Model for the NULB-T Problem

2.1 Assumptions

Assumptions for the NULB-T are of two types, those for individual U-lines and those for the JIT production unit.

For individual U-lines, the assumptions are those for the SULB-T problem:

1. Each U-line produces a single product.
2. Tasks are partially ordered by a set of precedence constraints, P.
3. All task times and task lengths are deterministic.
4. Task assignments to stations are limited by cycle time, precedence and location constraints.
5. Task distance, ℓ_x, consist of two parts; one part, $(1-\alpha_x)\ell_x$, is completed as part of task completion and the second part, $\alpha_x\ell_x$, represents travel between adjacent tasks. Only the second part, $\alpha_x\ell_x$, is assigned a travel time.
6. All distances are measured as Euclidean distances.
7. Travel time/unit distance is assumed to be identical for all operators.

For a JIT production unit the following assumptions are made:

8. There are N U-lines which may be considered for simultaneous balancing.
9. The N U-lines are located sufficiently close together that tasks from every U-line may be included in a multi-line station with tasks from at least one other U-line. In the general case, the location of the U-lines is not fixed.
10. All U-lines operate at the same cycle time. (The case where cycle times are not identical is dealt with in the discussion)
11. Access to the U-lines is restricted to the opening of each U. Stations are not permitted to cross paths and so the number of multi-line stations associated with any U-line exceed two.
12. A multi-line station can include tasks from at most k U-lines.

13. All travel distances between U-lines are known constants.

2.2 Notation

NULB-T problem variables.

N = The number of U-lines in the JIT production unit.

C = The cycle time.

M_{total} = The total number of stations required to complete all tasks on the N U-lines.

k = The maximum number of U-lines whose tasks may be included in a single multi-line station.

g = the travel time per unit of distance

Variables specific to each individual U-line, u, u = 1, ..., N

X_u = The number of tasks to be completed on U-line u.

$x(u)$ = Task x to be completed on U-line u. $x = 1, \ldots, X_u$.

$t_{x(u)}$ = The processing time for task x on U-line u.

$\ell_{x(u)}$ = The task distance on U-line u required by task x(u).

$\alpha_{x(u)}$ = The proportion of $\ell_{x(u)}$ not directly included in the performance of task x(u) and assigned a travel time.

S_u = $\{x(u) \mid x = 1,\ldots,X_u\}$, the set of tasks on U-line u.

P_u = The set of precedence constraints for S_u.

M^*_u = The optimal number of stations required to balance U-line u separately.

L_u = The length of each of the front and back of U-line u.

W_u = The width of U-line u.

We assume without loss of generality that U-lines are constructed of straight lines and right angles and that for each U-line u, $\sum_{x(u)\in Su} \ell_{x(u)} = 2L_u + W_u$.

$S\delta_{j(u)}$ = A feasible subset, j, of tasks from U-line u which may be assigned to

multi-line stations. Each $S\delta_{j(u)}$ is selected so that it is theoretically possible to balance S_u - $S\delta_{j(u)}$ in M^*_u-δ stations, with δ=1,2.

δ = the number of single line stations to be removed from a U-line and assigned to multi-line stations.

Variables specific to a station:

Station variables may relate either to a station containing tasks from a single U-line u or to a multi-line station including tasks from up to k different U-lines. A denotes the U-lines which contribute tasks to a station m(A), where A = u, uv, uvw, uvwx,.. ; u,v,w,... = 1,...,N and u≠v≠w≠x≠...

m(A) = Station m from U-line(s) A

M_A = The number of stations from U-line(s) A.

$ST_{m(A)}$ = {x(u) | task x(u) is performed at station m(A)}.

$DT_{m(A)}$ = Travel distance in station m(A) (excluding travel involved in task performance).

$g \times DT_{m(A)}$ = The time required to travel distance $DT_{m(A)}$.

2.3 NULB-T Model

Minimize $M_{total} = \sum_{A} M_A$,

Subject to:

$$\bigcup_{A} \left(\bigcup_{m=1}^{M_A} ST_{m(A)}\right) = \bigcup_{u=1}^{N} S_u, \qquad (1)$$

$$ST_{m(A)} \cap \underset{m(A)\neq m(A)'}{} ST_{m(A)'} = \varnothing, \quad \text{for all m(A)} \qquad (2)$$

$$\sum_{x(u)\in ST_{m(A)}} t_{x(u)} + g \times DT_{m(A)} \leq C, \quad \text{for all m(A)} \qquad (3)$$

$$M_{uv} + M_{uvw} + M_{uvwx} + \ldots \leq 2, \quad \text{for any u} \qquad (4)$$

Multi-line stations can contain tasks from at most k U-lines. (5)

Precedence constraints for each U-line. (6)

Location constraints (7)

The objective of the NULB-T problem is to minimize the total number of stations required to balance the JIT production unit. All tasks on the N U-lines are assigned to a station (1). Each task is assigned to only one station (2). The total task time plus travel time in each station cannot exceed the cycle time (3). Tasks from any U-line can be contained in at most two multi-line stations (4), while the maximum number of U-lines whose tasks may be included in a single multi-line station cannot exceed k (5). Location constraints (7) restrict the physical arrangement and interaction of both single line and multi-line stations.

3. A Solution Algorithm for the NULB-T Problem

The algorithm for solving the NULB-T problem consists of three steps.

Step 1: All N U-lines are balanced separately to find the total number of stations needed to operate the JIT production unit if no multi-line stations are used.

Step 2: From each U-line, u, a subset of tasks to be assigned to the multi-line stations is removed, reducing the number of single-line stations remaining on u. These subsets are called *multi-line subsets*. Up to N multi-line subsets may be selected, at most one from each U-line.

Step 3: The tasks in the multi-line subsets are assigned to multi-line stations in a way that

minimizes the total number of stations required to operate the JIT production unit.

The procedure for solving both the general and fixed location NULB-T problems is identical for the first two steps in the algorithm. This common procedure is contained in Section 3.1. Procedures for assigning tasks to multi-line stations are presented in section 3.2 for the general case and section 3.3 for the fixed location case. A detailed description of the algorithm follows.

3.1 Selecting Multi-line Subsets

Step 1. Balance all of the U-lines separately.

For each U-line, u, find M^*_u, the optimal number of stations required to balance u alone. The SULB-T branch and bound algorithm from of Part II of this thesis is used to solve the N individual SULB-T problems.

Step 2. Select a multi-line subset from each U-line.

Multi-line subsets from any U-line may take one of two forms, depending on whether we are trying to reduce the number of single line stations on the line by one or two (denoted by $\delta = 1$ or 2).

$S1_u$ - A subset which allows the remaining SULB-T problem, $S_u - S1_u$, to be solved using M^*_u-1 stations. This subset is assigned to one multi-line station or may be divided between two multi-line stations if it is a *crossover subset*, one with tasks on both the front and back of the U-line.

$S2_u$ - A subset which allows $S_u - S2_u$, to be balanced using M^*_u-2 stations. Including travel the subset time must exceed C and so the subset must be

divided between two multi-line stations. It is assumed that $S2_u$ must contain tasks on both sides of the U-line.

All feasible subsets $S\delta_u$, $\delta = 1,2$ and $u=1,...N$ are found separately using the following procedure.

Step 2.1 Bounds

Determine upper and lower bounds on the total task time plus travel time which may be assigned to each subset $S\delta_u$. These bounds are discussed in the Appendix.

Step 2.2 Enumeration of potential multi-line subsets

Enumerate all candidate subsets falling within the bounds. The candidates are denoted as $S\delta_{j(u)}$, denoting the jth subset on U-line u which could potentially remove δ single line stations from u. Not all U-lines will have feasible candidate subsets for $\delta=1,2$.

Step 2.3 Selection of multi-line subsets, $S\delta_u$,

The algorithm searches for two smallest subsets $S\delta_{j(u)}$, $\delta=1,2$ which allow the remaining ULB problems S_u-$S\delta_{j(u)}$ to be solved using M^*_u-δ stations. The procedure is:

Subset Selection Loop

Select a candidate subset $S\delta_{j(u)}$

Solve SULB-T problem, S_u-$S\delta_{j(u)}$ to find M.

IF $M > M^*_u - \delta$ THEN fathom $S\delta_{j(u)}$ and any $S\delta_{g(u)} \subset S\delta_{j(u)}$

IF $M = M^*_u - \delta$ THEN fathom any $S\delta_{g(u)}$ with time, $t(S\delta_{g(u)}) \geq$

$t(S\delta_{j(u)})$ and denote $S\delta_{j(u)}$ as the current solution $S\delta_u$.

Loop until all subsets are fathomed leaving the best $S\delta_u$.

A binary selection procedure is used to reduce the number of subsets tested.

Repeat the procedure for $u=1,\ldots,N$ and $\delta=1,2$ until all feasible multi-line subsets $S\delta_u$ have been identified. There will be a maximum of 2N subsets $S\delta_u$. Although the subset time for $S\delta_u$ is the smallest possible there is no guarantee that some fathomed subset $S\delta_{g(u)}$ with greater time might not have contained a better mixture of tasks for combining with other subsets. Consequently, the algorithm proposed is a heuristic.

Step 2.4 Selection of a single multi-line subset from each U-line

From each U-line select the subset $S\delta_u$ with the maximum idle time.

The problem of assigning the tasks in multi-line subsets to stations will be called the *multi-line station assignment problem.* Proving an optimal solution would require testing every possible combination of feasible subsets $S\delta_{j(u)}$ in the N U-lines. Even if it is assumed that the best $S\delta_u$ was selected, given a set of multi-line subsets, $S\delta_u$, $\delta = 1,2$ and $u = 1,\ldots,N$, proving the optimal multi-line station assignments would still require solution of a multi-line station assignment problem for every possible combination of the subsets, up to 2^N problems. The multi-line station assignment problem will be shown to be NP-hard so this is a significant task. The decision on how many such problems to attempt depends on the size of the problem, the importance and value the solution and the computing resources available. In this paper, to reduce the problem to a manageable level, only one

subset is selected from each U-line and a single multi-line station assignment problem is solved.

The procedure utilized to solve the multi-line station assignment problem depends on whether U-line locations are restricted or not. In section 3.2, the general case will be examined followed by the case where U-line locations are fixed in section 3.3.

3.2 The Multi-line Station Assignment Problem - General Case

The formulation for the multi-line station assignment problem is the similar to that for the NULB-T problem except that all single line stations have already been constructed and only the tasks in multi-line subsets remain to be assigned. All stations built from multi-line subsets will be referred to as multi-line stations even though some may ultimately contain tasks from only one U-line. The problem to be solved is:

Minimize $M_{multi} = \sum_{A} M_A$,

Subject to:

$$\bigcup_{m=1}^{M_A} ST_{m(A)} = \bigcup_{u=1}^{N} S\delta_u, \qquad i=1 \text{ or } 2 \tag{8}$$

$$ST_{m(A)} \cap \underset{m(A)\neq m(A)'}{} ST_{m(A)'} = \varnothing, \tag{9}$$

$$\sum_{x(u)\in ST_{m(A)}} t_{x(u)} + g(DT_{m(A)}) \leq C, \quad \text{all } m(A) \tag{10}$$

$$M_{uv} + M_{uvw} + M_{uvwx} + \ldots \leq 2, \text{ for any } u \tag{11}$$

Each multi-line station contains tasks from at most k U-lines (12)

Precedence constraints for the tasks in the multi-line subsets (13)

Location constraints on U-lines, paths and tasks (where applicable) (14)

where M_{multi} = The total number of stations formed from the multi-line subsets.

In the general case of the multi-line station assignment problem, U-line locations are determined after the station assignments are made. This could apply during facility design or if machines are totally flexible and each U-line is capable of handling any set of tasks S_u. The problem bears a resemblance to the bin-packing problem, in that all multi-line subsets must be packed into the minimum number of bins (stations) of size C. However, a number of important differences complicate the problem.

i. Tasks assigned to each multi-line station can come from at most k U-lines.

ii. As tasks are added to each multi-line station, a time for operator travel is also added. Thus, total problem time is a variable dependent on the multi-line station assignments.

iii. Since subsets may be divided the total number of subsets to be assigned to stations, #(S), varies depending on the type of subsets in the problem, their composition and whether they are divided.

The bin-packing problem may be shown to be a special case of the multi-line station assignment problem. In the multi-line station assignment problem, if constraint 12 is relaxed to k=#(S), travel time is assumed to be constant per station in constraint 10 and subsets are divided prior to solution - that is, #(S) is fixed - the problem becomes a bin-packing problem. The bin packing problem is NP hard and so the multi-line station assignment problem is also NP hard.

Step 3. Solving the Multi-Line Station Assignment Problem - General Case

Once the multi-line subsets have been selected, solving the multi-line station assignment problem involves assigning the tasks in those subsets to stations. It is assumed that minimizing operator travel, particularly between U-lines, and minimizing potential interference are secondary objectives to minimizing the number of stations. The following priority ranking is made concerning the desirability of potential multi-line station assignments. Provided that the total number of stations is not increased,

1. the preferred solution would be one in which multi-line subsets from individual U-lines are not divided,
2. the next would be a solution in which subsets from a single U-line are divided into one subset containing tasks from the front of the U-line and another containing tasks from the back of the U-line and
3. the least attractive solution would be one where multi-line subsets with tasks on both the front and back of the U-lines are divided in some other fashion, with one of the subsets containing tasks on both sides of the U-line.

The following heuristic algorithm for solving the multi-line station assignment problem considers the preference ranking above and uses three heuristics successively, stopping if one of the solutions achieves the lower bound on M_{multi}, the number of multi-line stations.

Step 3.1 Lower Bounds On M_{multi}

Two lower bounds are calculated, with the maximum used as $LB(M_{multi})$.

The first bound is based on the total task time plus a lower bound on travel time. The second considers, #(half), the number of indivisible subsets or tasks in other subsets whose total time is such that no two may be added to the same multi-line station. Therefore,

$$LB(M_{multi}) = Max\{ \lceil [\sum_{x(u)\epsilon S\delta_u,\ \delta=1 \text{ or } 2,\ u=1,\ldots,N} t_{x(u)} + g \times LB(DT_{multi})]/C \rceil,\ \#(half) \}$$

where, $LB(DT_{multi})$ is a lower bound on the travel within multi-line subsets and between U-lines required to achieve $LB(M_{total})$,

#(half) = the number of non-crossover subsets with total time $> [C-g \times d_{uv}]/2$ plus the number of tasks in other subsets with processing times plus travel time $> [C-g \times d_{uv}]/2$ and

d_{uv} = the minimum travel distance between any two U-lines. d_{uv} is assumed to be constant in the general problem. For the fixed location problem $d_{uv} = \min d_{st}$, $s \neq t$.

Step 3.2 Heuristic 1

H1.1 Divide all $S2_u$ subsets into two parts, if possible into one subset with tasks on the front of the U-line and one with tasks on the back. Do not divide $S1_u$ subsets.

H1.2 A single-pass heuristic with priority based on non-increasing subset times is used to assign the subsets to stations. As subsets are added, travel within the station is calculated, a travel time is assigned to the station and feasibility with respect to cycle time and location constraints is ascertained.

If $M_{multi} = LB(M_{multi})$ stop, otherwise continue to 3.3.

Step 3.3 Heuristic 2

H2.1 Without changing existing $S2_u$ divisions, divide all $S1_u$ subsets with tasks on both sides of the U-line into two subsets, one containing only tasks at the front of the U-line and one containing tasks at the back of the line.

H2.2 Solve the multi-line station assignment problem with the new list of subsets using the heuristic in 3.3.2.

If $M_{multi} = LB(M_{multi})$ then stop, otherwise continue to 3.4.

Step 3.4 Heuristic 3

H3.1 Calculate I_{max}, the maximum average idle time per station which will allow attainment of the lower bound on M_{multi}.

$$I_{max} = [C \times LB(M_{multi}) - \sum_{u=1}^{N} \sum_{x(u) \in S\delta_u} t_{x(u)} + g \times LB(DT_{multi})\ ] / LB(M_{multi})$$

H3.2 For each $S\delta_u$ with tasks at both ends of the U-line, enumerate and store every feasible division of the subset into two parts $S\delta^f_{j(u)}$ and $S\delta^b_{j(u)}$. Designate all non-crossover subsets which cannot be divided as $S1^{nc}_u$.

H3.3 Assign subsets and tasks to multi-line stations using the following procedure:

Set $m = 1$, $I_m = C$.

1. From the set of available subsets, $S1^{nc}_u$ with $t(S1^{nc}_u) < I_m$, select the subset with max $t(S1^{nc}_u)$ and add it to station m. Update station travel distance, idle time and the number of U-lines with tasks assigned to station m: DT_m, I_m, and $\#(u)_m$. If the station is infeasible with respect to cycle time or location

constraints then remove $S1^{nc}_u$ and select the subset with the next greatest time, otherwise continue.

- Repeat step 1 until $nu_m = k$ or there is no $S1^{nc}_u$ which fits in the current station.
- If there are no unassigned $S1^{nc}_u$ subsets proceed to step 5.

2. Calculate the average accumulated idle time per station created, I_{avg}.
   - If $I_{avg} \leq I_{max}$ go to step 4.
   - If $nu_m < k$ and $I_{avg} > I_{max}$ then proceed to step 3.
   - If $nu_m = k$ and $I_{avg} > I_{max}$ then determine if an available $S\delta^f_{j(v)}$ or $S\delta^b_{j(v)}$ can reduce I_{avg} if it replaced the last subset added to m. If there is such a $S\delta^f_{j(v)}$ or $S\delta^b_{j(v)}$ then backtrack by removing the last subset added to the station and proceed to step 3, otherwise move on to step 4.
3. Select the available $S\delta^f_{j(u)}$ or $S\delta^b_{j(u)}$ with largest time including travel $\leq I_m$ and add it to the current station. If the station is feasible then the complement of that subset is now indivisible and is designated as $S1^{nc}_u$ for step 1. Return to step 2. If the station is not feasible then fathom the subset and repeat step 3. If no subset fits then proceed to step 4.
4. Create a new station; $m=m+1$, $I_m = C$ and $\#(u)_m = 0$. Return to step 1.
5. If all $S1^{nc}_u$ subsets have been assigned, begin assigning $S2^f_{j(u)}$ or $S2^b_{j(u)}$ subsets in step 1, selecting the largest which fits into the current station. Its complement becomes a $S1^{nc}_u$ subset. Finally, if there are no available $S2^f_{j(u)}$

or $S2^{b}_{j(u)}$ subsets then $S1^{f}_{j(u)}$ or $S1^{b}_{j(u)}$ subsets are chosen.

Note that the optimal solution to the problems solved by heuristics 1 and 2 could be found using a modified version of the SULB-T algorithm treating each subset as an individual task to be assigned. However, since heuristic 3 offers more flexibility and a greater opportunity to the lower bound the algorithm proceeds immediately to it.

Example 1. An example of the general case multi-line station assignment problem.

Consider a NULB-T problem with the following parameters: N=7, C=370 seconds, k = 3, g = 0.3 sec/ft, W_u= 14 feet for all u, d_{uv}= 20 feet for all uv and $\alpha_{x(u)}$ = .8 for all tasks. The precedence graphs for the U-lines are shown in Figure 3. To simplify the example only subsets $S1_u$ are considered.

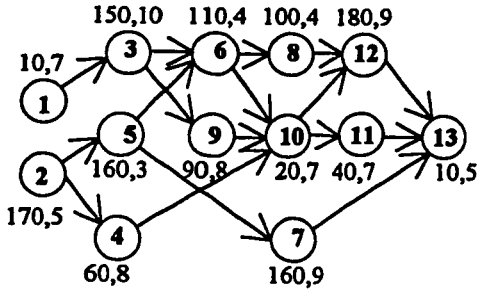
Step 1. Each of the U-lines is solved separately. 28 stations are required to balance the JIT production unit.

Step 2. All potential multi-line subsets $S1_{j(u)}$ are enumerated. The subset $S1_{j(u)}$ with the smallest total time which allows S_u- $S1_u$ to be solved in M^{*}_{u}-1 stations is denoted as $S1_u$.

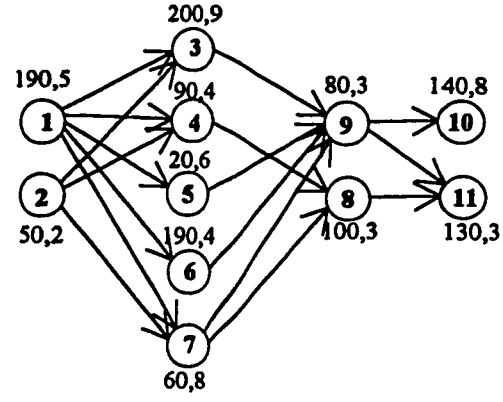
The results from the first two steps of the NULB-T algorithm are shown in Table 1. All tasks except those in the multi-line subsets $S1_u$ have been assigned to 21 single-line stations. The multi-line station assignment problem is to assign the tasks from the 7 $S1_u$ subsets shown in Table 1 to multi-line stations.

Step 3. The procedure described in section 3.2 is used to assign the multi-line subsets to

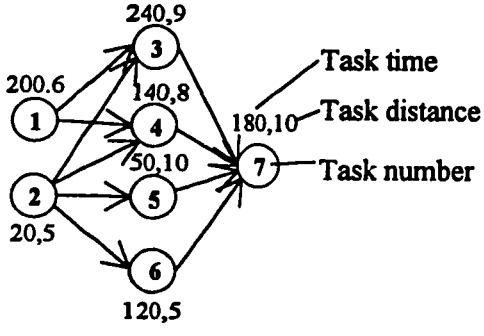
Figure 3
Precedence Graphs for Example 1



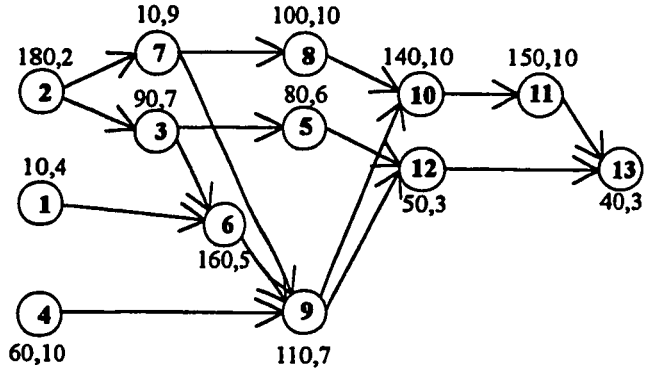
U-line 1



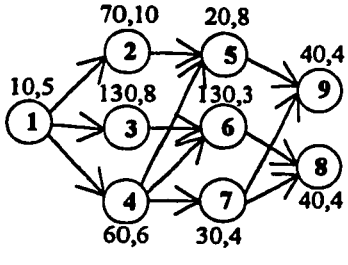
U-line 5



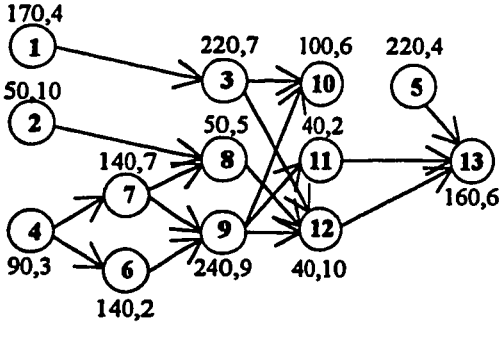
U-line 2



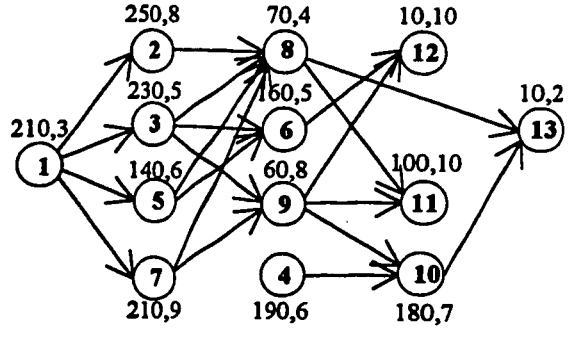
U-line 6



U-line 3



U-line 4



U-line 7

Table 1

Results from Steps 1 and 2 of the NULB-T Algorithm

U-line	# of tasks, X_u	M^*_u	# of $S1_{j(u)}$	# SULB-T solved to identify $S1_u$	Multi-line subset, $S1_u$	Total Subset time
1	13	4	37	6	{1,2,13}	210.0
2	7	3	7	4	{2,5,7}	259.6
3	9	2	85	8	{1,4,5,8,9}	179.7
4	13	5	37	10	{11,12,13}	249.7
5	11	4	15	4	{1}	192.7
6	13	4	49	5	{4,12,13}	157.1
7	13	6	32	8	{4}	193.2
Total	79	28	262	45	19	1442.0

stations.

Step 3.1 Lower bounds on M_{multi}: At least $\lceil 1442/370 \rceil = \lceil 3.898 \rceil \rightarrow 4$ stations are required. Since there are seven U-lines with feasible multi-line subsets, at least three of the subsets must be combined with subsets from other lines to achieve this lower bound. With k = 3 there will be a minimum of five traverses between U-lines and so $5 \times g \times d_{uv} = 5 \times .3 \times 20 = 30$ seconds is added to the sum of the subset times.

$LB1(M_{multi}) = \lceil 1472/370 \rceil = \lceil 3.98 \rceil = 4.$

For the second lower bound, travel time and return between U-lines requires 2×20 ft $\times$ .3 sec/ft. = 12 seconds. If a non-crossover subset or a task plus its travel in another subset has total time > (370 -12)/2 = 179 it will not fit into a station with another such non-crossover subset or task. Non-crossover subsets $S1_4$, $S1_5$ and $S1_7$ and tasks 12 on U-line 1 and 7 on U-line 2 all have total time > 179 so $LB2(M_{multi}) = 5$.

$\therefore LB(M_{multi}) = 5 \rightarrow I_{max} = (5 \times 370 - 1472)/5 = 75.6$ seconds per station.

Step 3.2 Heuristic 1: The undivided $S1_u$ subsets are assigned to stations using Heuristic 1. The resulting assignment of tasks to six stations are shown in Table 2. A single station has been removed but the lower bound on multi-line stations has not been achieved so the algorithm proceeds to step 3.3.

Step 3.3 Heuristic 2: All crossover subsets are divided into front and back subsets. Heuristic 2 is used to assign the resulting eleven subsets to five stations as shown in Figure 4. This solution achieves the lower bound of five stations and the

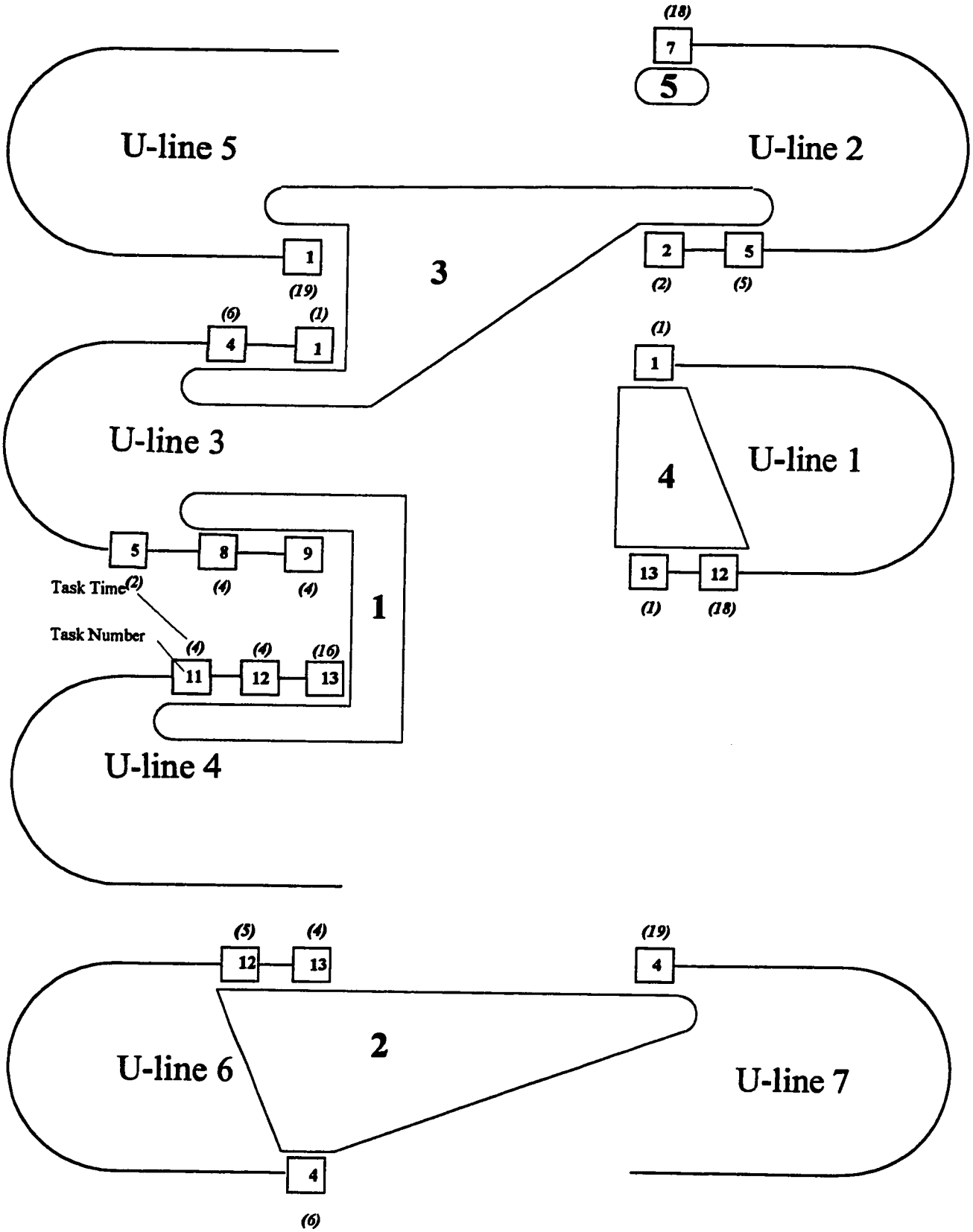
Table 2

Multi-line Station Assignments After Heuristic 1.

Station	Subsets	Station time
1	$S1_2$	259.6
2	$S1_4$	249.7
3	$S1_1$	210.0
4	$S1_7$,$S1_6$	362.3
5	$S1_5$	192.7
6	$S1_3$	179.8

Figure 4

Example of a General NULB-T Balance with Divided Subsets



algorithm halts. The U-lines are arranged as dictated by the solution. Twenty-six stations are needed to operate the seven U-lines in the facility, a 7.1% reduction in the total labour requirement.

3.3 The Multi-line Station Assignment Problem - Fixed U-line Locations

When the location of U-lines is fixed, tasks from any U-line may be combined with those from only a limited number of nearby U-lines into multi-line stations. Four factors take on increased significance in the fixed location problem.

1) The number of U-lines close enough to contribute tasks to a multi-line station with U-line u.
2) The actual distance between U-lines: Actual distances between U-lines are measured from the centre of the U-line openings and stored in an N×N array.
3) The feasibility of combining tasks from U-line pairs: Because of the layout many pairs of U-lines may not be included in the same station. Such pairs are assigned large separation distances which act to prohibit these pairs.
4) The paths between U-lines and interference between paths: Paths between U-lines are drawn as one or a series of straight lines between the centre of each U-line opening. A prospective path must be compared to existing paths between other pairs to ensure interference does not occur.

Step 3. Solving the Multi-Line Station Assignment Problem - Fixed Location Case

When U-line locations are fixed, step 3 of the NULB-T algorithm is altered to

reflect the location restrictions placed on the solution. As in the general case, the objective is to achieve the lower bound on M_{multi}. The procedure begins with the station with the least number of options, a non-crossover subset with the fewest potential U-lines with which to share a station.

Step 3.1 Lower Bounds on M_{multi}

As in section 3.2, calculate the lower bound on the number of multi-line stations, $LB(M_{multi})$, and the maximum accumulated idle time which will achieve the bound, I_{max}.

Step 3.2 Fixed location multi-line station assignment heuristic

Assign the subsets to stations using heuristic 4.

H4.1 Select the first $S1^{nc}_u$ subset from those which fit into the current multi-station using the following priorities.

- Select the subset with the least number of available adjacent U-lines. A U-line is considered available if it has unassigned tasks.
- Break ties by selecting the subset with the largest subset time. If no $S1^{nc}_u$ subsets are available then proceed to step H4.7.

H4.2 If the first subset is taken from U-line v select the next subset with total time $\leq I_m - d_{uv}$ from the U-lines adjacent to v using the following priorities.

- Subsets are preferred in order:

  $S1^{nc}_u > S2^{f}_{j(u)}, S2^{b}_{j(u)} > S1_u, S1^{f}_{j(u)}, S1^{b}_{j(u)}$.

- From the preferred subsets select the subset with the least number of available partner U-lines.

· Break ties by selecting the subset with total time $\leq I_m - d_{uv}$.

H4.3 If the station contains two or more U-lines plot the station path(s) between the U-lines.

· Check for interference by determining whether a current path between a pair of U-lines uv intersects an existing multi-line station path. If this happens the pair uv is infeasible. Remove the last subset added and add a large number to the cell uv in the distance array to prohibit the pair. Return to step H4.2.

H4.4 Update the idle time, I_m, and the average accumulated idle time, I_{avg}.

· If $I_{avg} \leq I_{max}$ go to step H4.6.

· If $\#(u)_m < k$ and $I_{avg} > I_{max}$ then proceed to step H4.5.

· If $\#(u)_m = k$ and $I_{avg} > I_{max}$ then determine if an available $S\delta^f_{j(u)}$ or $S\delta^b_{j(u)}$ can improve the solution over the last subset added. If there is such a $S\delta^f_{j(u)}$ or $S\delta^b_{j(u)}$ then backtrack by removing the last subset added in step H4.2 and proceed to step H4.5, otherwise proceed to H4.6.

H4.5 Select the available $S\delta^f_{j(u)}$ or $S\delta^b_{j(u)}$ with largest time plus travel $\leq I_m$ and add it to the current station. The complement of that subset now becomes indivisible and is designated as $S1^{nc}_u$ for step H4.1. Return to step H4.3.

H4.6 Create a new station; m=m+1, $I_m = C$ and $\#(u)_m = 0$. Return to step H4.1.

H4.7 As described in the general case, once all $S1^{nc}_u$ subsets have all been assigned select $S2^f_{j(u)}$ and $S2^b_{j(u)}$ subsets and then $S1^f_{j(u)}$ and $S1^b_{j(u)}$ subsets.

H4.8 If $M_{total} > LB(M_{total})$ then any subsets or stations with #(u) < k may be considered for inclusion in stations with non-adjacent U-line subsets where cycle time and location constraints permit.

Figure 5 shows a solution to the fixed location problem for Example 1 where U-line locations are fixed. The solution achieves the lower bound of five stations.

3.4 A Comparison of Complexity of Line Balancing Problems

Although all line balancing problems are NP-hard, the computational complexity involved in enumerating all feasible subsets varies depending on the particular problem. For NULB-T problems the major computational effort required is for solving repeated individual SULB-T problems to identify the multi-line subsets $S\delta_u$. Once these subsets are selected the multi-line station assignment problem must then be solved. The limit of k U-lines per multi-line station means that the complexity of this problem is O(N!/(N-k)!k!). A comparison of the computational complexity of the different types of line balancing problems discussed in this paper is found in Table 3.

4. A Computational Study of the NULB-T Problem

A computational study was undertaken to investigate the benefits of multiple U-lines and the effectiveness of the solution algorithms.

4.1 Study Parameters and Procedure

100 general NULB-T problems were generated and solved as follows.

1) N ~ U[3,9].

Figure 5

Example of a NULB-T balance with fixed locations

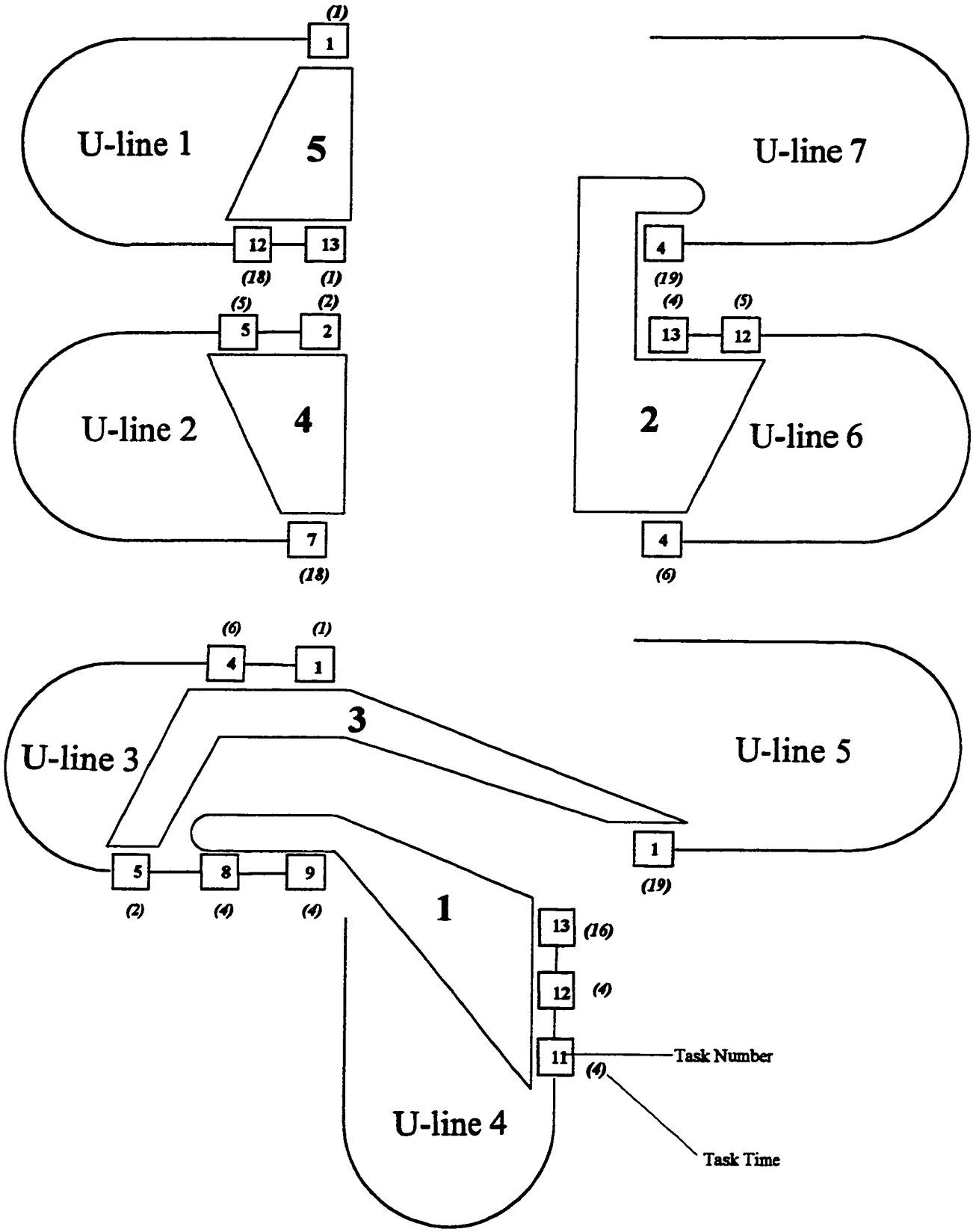


Table 3

Computational Complexity of Different Line Balancing Problems

Problem	Computational Complexity	References
SALB	$O(N_u^2 \#(S^f_u))$	Queyranne [1985]
SULB	$O(N_u^2 (\#(S^f_u))^2)$	part I of this thesis
SULB-T	$O(N_u^2 (\#(S^f_u))^2)$	part II of this thesis
2ULB-T	$O(\#(S_{multi(1)})[N_1^2 (\#(S^f_1))^2] + \#(S_{multi(2)})[N_2^2 (\#(S^f_2))^2])$	part II of this thesis
NULB-T	$O(\sum_{u=1,..,N} \#(S_{multi(u)})[N_u^2(\#(S^f_u))^2]) + O(N!/(N-k)!k!)$	part III of this thesis

Notes $\#(S^f_u)$ = the number of forward feasible subsets from U-line u.

$\#(S_{multi(u)})$ = the number of feasible multi-line subsets from U-line u.

2) N U-lines were randomly selected from a problem set of 30 U-lines, each containing 7-18 tasks. See Appendix 2 for a description of the problem set.

3) Cycle time for each problem was randomly selected to be between the maximum task time in the problem + 10 seconds (to allow for travel) and twice the maximum task time.

4) Only $S1_u$ multi-line subsets were considered in the algorithm to simplify both programming and computation.

5) For all problems k = 3. This is a reasonable limit for travel between different U-lines.

6) For any single U-line a maximum of 200 subsets $S1_{j(u)}$ was enumerated. This limit was reached in 3.3% of the 604 U-lines whose subsets $S1_{j(u)}$ were enumerated. It is unlikely that increasing the maximum beyond 200 will significantly improve the final solution.

7) Two criteria were tested for selecting subset $S1_u$ from the set of multi-line subsets. Define $S_{multi(u)} = \{S1_{j(u)} \mid S_u\text{-}S1_{j(u)}$ is balanced with M^*_u-1 stations$\}$, the set of subsets which allowed a single-line station to be removed from u. The two criteria used were:

Criterion 1. Select the subset with the smallest subset time, $t(S1_{j(u)})$.

Criterion 2. Identify the subset, $S1_{j(u)}$, with the smallest subset time and the crossover subset, $S1_{c(u)}$, with the smallest subset time where $S1_{j(u)}, S1_{c(u)} \in S_{multi(u)}$.

$$\text{then if } \begin{cases} j = c & \rightarrow S1_u = S1_{j(u)} \\ \text{if } t(S1_{j(u)}) + 20 \text{ sec.} \geq t(S1_{c(u)}) & \rightarrow S1_u = S1_{c(u)} \\ \text{if } t(S1_{j(u)}) + 20 \text{ sec.} < t(S1_{c(u)}) & \rightarrow S1_u = S1_{j(u)} \end{cases}$$

Thus, if a non-crossover subset $S1_{j(u)}$ has the smallest time but a crossover subset $S1_{c(u)}$ is within 20 seconds of the total time then the more flexible crossover subset, $S1_{c(u)}$ is selected. Twenty seconds is approximately 10% of the average maximum task time for the problem set.

8) The algorithm for solving the general case of the NULB-T problem was used as described in sections 3.1 and 3.2 (with $S1_u$ subsets only). Because only $S1_u$ subsets were considered, a third lower bound on M_{multi} was added, $LB(M_{multi}) = \lceil \#(S1_u)/k \rceil$.

9) The multi-line station assignment problem was solved using heuristics, H1, H2 and H3.

H1 - The problem with undivided subsets $S1_u$ was solved assigning tasks with priority based on non-increasing subset times.

H2 - Any crossover subset, $S1_u$, was divided into front and back subsets, $S1^f_u$ and $S1^b_u$, and the problem was solved using the same heuristic as in H1.

H3 - The heuristic was used as described in section 3.4.

Programs were coded in QuickBasic and run on a 50 mhz 486 microcomputer.

4.2 The Study Results

The results of the study of 100 NULB-T problems are summarized in Table 4. Several results are worthy of note.

1. The average NULB-T problem contained 6.04 U-lines but only 5.76 had feasible

Table 4

Selected Results for 100 NULB-T Problems Solved Using Criterion 1.

# of U-lines	ΣM^{*}_{u} (sum of independent solutions)	# of multi-line subsets, $S1_{u}$	# of stations eliminated from M_{total}	% decrease in M_{total}	NULB-T solution time (sec.)
Min: 3	Min: 8	Min: 0	Min: 0	Min: 0.0	Min: 51
Max: 9	Max: 38	Max: 9	Max: 5	Max: 33.3	Max: 597
Mean:**6.04**	Mean: **19.9**	Mean: **5.76**	Mean:**2.09**	Mean:**10.6**	Mean: **267**

multi-line subsets. On average the selection process involved solving 5.97 of the possible 53.3 $S1_{j(u)}$ subsets enumerated to select $S1_u$. In the final solution an average of 2.09 stations were eliminated, resulting in a mean reduction in the number of stations of 10.59 % over separate solution of the U-lines.

2. The problems are difficult, typically taking about 4.5 minutes to solve. The reason is obvious when we consider that to solve the average N U-line problem with 6 U-lines the solution algorithm had to proceed through the following average process.

   Step 1. Solve 6 SULB-T problems with S_u to find all M^*_u.

   Step 2. Enumerate an average of 53 potential multi-line subsets $S1_{j(u)}$ per U-line and solve 6*6 = 36 SULB-T problems of S_u- $S1_{j(u)}$ to identify the best multi-line subsets $S1_u$.

   Step 3. Solve the multi-line station assignment problem.

3. In the multi-line station assignment problem the three heuristics used performed effectively.

   H1 achieved LB(M_{multi}) in 67 of the 100 problems and the algorithm halted.

   H2 achieved the lower bound in 10 of the remaining 33 problems but improved the solution over H1 in 13 problems.

   H3 improved the solution over H2 in 9 of the 23 problems and reached the lower bound in each of the nine problems.

   Used together the three heuristics achieved the lower bound on multi-line stations in 86% of the problems and obtained the optimal solution in 97% of the problems.

4. If criterion 2 is used to select $S1_u$ subsets, the results are very similar. For the entire problem set criterion 2 requires 1 more station than criterion 1. Using both criteria together and solving all problems twice if the lower bound is not reached reduces the total number of stations by 2, a decrease of 0.04% in M_{total}.

5. Discussion

Previous research has shown U-lines to be more efficient than comparable straight production lines. Simultaneous balancing of multiple U-lines increases this productivity advantage considerably. However, implementation of N U-line balancing requires consideration of several factors. Operators must be trained to complete many tasks, not just those on a single U-line. This is a more reasonable undertaking in the fixed location problem where only a limited number of the tasks are feasible for any station. Material handling methods and paths must support multi-line stations.

Numerous extensions to the problem exist. One which will appear in many facilities is the problem of balancing multiple lines where the cycle times are not identical. Although the case of unrelated cycle times is beyond the scope considered in this paper, the situation where cycle times are integer multiples of the smallest cycle time may exist where different parts for the final product are required in different quantities. The problem may be solved as described in section 3 using the smallest cycle time, C_{min}, for the multi-line station assignment problem. In implementing the solution, each U-line will be visited only once every cycle, C_u. This will be once every C_u/C_{min} of the C_{min} cycles.

5.1 Extensions to NULB-T research

Solving the NULB-T problem is a complex process requiring substantial time and memory. Numerous opportunities exist for future research into both the general and fixed location problems including:

An investigation of means of improving solution algorithms through

· Improved bounds on travel distances and on the number of multi-line stations.

· Methods of reducing the number of multi-line subsets evaluated to find the best $S\delta_{j(u)}$.

· Identifying the best criteria for selecting multi-line subsets $S\delta_u$ and for choosing between subsets $S1_u$ and $S2_u$, where both exist.

Selection criteria for the final subset may be based on some combination of the following factors:

· Idle time in each of the subsets,

· The number of tasks/minimum number of stations,

· The distribution of tasks and task times at the front and back of the line.

· The consideration of more than one multi-line subset from each line for the multi-line subset assignment problem.

· The use of other heuristics to solve the problem.

· Integer programming approaches to the problem.

Improved heuristics for the fixed location problem

The heuristics proposed for solving the fixed location multi-line station assignment problem examine only a small portion of the solution space. The final solution is

determined by the initial starting point and by the rules for adding tasks to the station. Other algorithms which use different rules for determining starting location and selecting the next subset or tasks to be added to a station should be considered. Statistical search techniques such as simulated annealing may prove useful for this problem.

Allowing unrestricted access to U-lines

Removing the restriction on access to the U-lines dramatically increases flexibility in multi-line station assignment and, with it, the size of the solution space. An investigation of heuristic algorithms for the NULB-T problem where all or selected sections of U-lines are available for multi-line station assignment would examine the effect of a preference hierarchy for potential solution and combinations of heuristic strategies which could be employed.

Extending the problem to other manufacturing layouts including lines with other shapes

Many advantages of multiple U-line balancing may be obtained to varying degrees in other line configurations. If segments of two or more production lines are situated close together it may be possible to apply multiple U-line balancing techniques to them to capture some of the advantages of multiple line balancing. Many of these areas are topics of ongoing research.

If the U-line problems are expanded to include multiple product lines and stochastic processing times the N U-line balancing problem is complicated even further. An extensive investigation into predominantly heuristic approaches to solving these problems is essential to further research into U-line balancing. The productivity benefits of multiple

U-line balancing and the success of even simple heuristics in attaining these gains provide a strong incentive for continuing this research.

References

Hackman, S.T., M.J. Magazine and T. S. Wee, "Fast, Effective Algorithms For Simple Assembly Line Balancing Problems," *Oper. Res.*, 37, 6, 916-924 (1989).

Monden, Y., *Toyota Production System*, Second Edition, Industrial Engineering Press, Institute of Industrial Engineers, Norcross, GA (1993).

Queyranne M., "Bounds for Assembly Line Balancing Heuristics," *Oper. Res.*, 33, (1985), 1353-1359.

Schonberger, R.J., *Japanese Manufacturing Techniques, Nine Hidden Lessons in Simplicity*, The Free Press, New York, 1982.

Van Assche, F. and W.S. Herroelen, "An Optimal Procedure for the Single-Model Deterministic Assembly Line Balancing Problem," *Eur. J. of O.R.*, 3, 142-49 (1978).

Wantuck, K.A., *Just In Time for America*, The Forum Ltd., Milwaukee, WI, 1989.

Appendix

Bounds on the sum of task times in a multi-line subset

The algorithm removes a subset of tasks from a single line to be assigned to multi-line stations. A feasible subset $S\delta_{j(u)}$ must have the potential to reduce the number of single line stations by δ and must meet the following conditions:

1) It must be theoretically possible to solve the problem S_u-$S\delta_{j(u)}$ in M^*_u-δ stations.

2) At least one task must be removed from S_u and assigned to $S\delta_u$.

3) There must be sufficient idle time in $S\delta_u$ to accommodate at least one task from another U-line and travel to the nearest U-line.

4) Tasks assigned to $S\delta_u$ must also obey the precedence relationship P_u.

Based on these conditions upper and lower bounds on multi-line subset time, $t(S\delta_{j(u)})$, may be determined as:

$$LB[t(S\delta_{j(u)})] = Max\{ \sum_{x=1}^{N_u} t_{x(u)} + g \times LB(DT_u) - (M^*_u - \delta)C, \min_{\text{available } x(u)} \{t_{x(u)}\} \},$$

$$UB[t(S\delta_{j(u)})] = C - \min_{\text{available } x(v),\ v \neq u} \{t_{x(v)}\} + g \times \min d_{uv}$$

where

- task x(u) is available if it has no predecessors or successors

- min d_{uv} is the minimum separation distance between all pairs of U-lines, uv and

- $LB(DT_u)$ is the lower bound on the travel distance in stations on U-line u.

$$LB(DT_u) = \sum_{x=1}^{N_u} \ell_{x(u)} + W_u$$

Subsets $S2_{j(u)}$ must be divided between two multi-line stations and must meet one

other condition in addition to those cited above. Because access to the U-lines is restricted to the opening of the U, $S2_{j(u)}$ subsets must include task at both the front and back of U-line u.

Enumerating all subsets of tasks from each U-line which could be assigned to $S\delta_{j(u)}$.

All feasible subsets on each U-line which could be assigned in the multi-line station are enumerated using the procedure described in Part II of this thesis. These subsets are feasible with respect to P_u and satisfy the bounds on total task time computed previously. That is:

$$UB[t(S\delta_{j(u)})] \geq \sum_{x \in S\delta_{j(u)}} t_{x(u)} \geq LB[t(S\delta_{j(u)})]$$

Mixed-Model Line Balancing:

U-Lines and Smoothing Station Assignments

Abstract

Many manufacturers are switching their production lines from single product or batch production to *mixed-model* production, often as a consequence of implementing just-in-time (JIT) principles into their operations. In mixed-model production, different products or models are produced on the same line with the models interspersed throughout a production sequence. This helps manufacturers provide their customers with a variety of products in a timely and cost-effective manner. The mixed-model U-line balancing (MMULB) problem assigns the tasks required to produce all models to a minimum number of stations on a U-shaped line. U-lines are widely used and well-suited to mixed-model production. A model of the MMULB problem is developed in this paper. The problem is NP-hard. An approximate solution algorithm is presented, and an illustrative example is worked. Areas where more research is needed are identified.

This paper has been submitted to the International Journal for Production Research.

1. Introduction

Many manufacturers are switching their production lines from single product or batch production to *mixed-model* production, often as a consequence of implementing just-in-time (JIT) principles into their operations. In mixed-model production, different products or models are produced on the same line, with the models interspersed throughout a production sequence. This allows manufacturers to provide their customers with a variety of products in a timely and cost effective manner. Successful mixed-model production requires solving the following problems.

1. *The mixed-model line balancing problem:* How will production tasks be assigned to stations on the line?
2. *The mixed-model sequencing problem:* In what sequence will units of different models be produced on the line?

This paper investigates the first problem, particularly as it relates to U-lines in JIT production systems. We will see that U-lines are well suited for mixed-model production and offer significant advantages over traditional straight lines.

Straight Line Literature: A large literature exists on the single-model line balancing problem for straight lines. See, for example, Baybars [1986], Talbot et al. [1986], Ghosh and Gagnon [1989], Hackman, Magazine and Wee [1989], Hoffmann [1992] and Johnson [1988]. Thomopolous [1967, 1970] and Macaskill [1972] were first to study the mixed-model line balancing problem for straight lines. The mixed-model sequencing problem for straight lines has been studied by Bard et al. [1992], Yano and Rachamadugu [1991],

and others.

U-line Literature: A small but growing literature exists on the single-model U-line balancing problem. See, for example, Miltenburg and Wijngaard [1994] and Parts I to III of this thesis. To date, no one has studied the mixed-model U-line balancing problem. The mixed-model sequencing problem (for U-lines) in JIT production systems is analysed in Hall [1983], Kubiak [1993], Miltenburg and Goldstein [1991] and Monden [1993].

The nature of mixed-model production presents additional challenges for line balancing compared to single-model production. In both cases, the line balancing problem seeks to minimize the number of stations (or maximize a measure of balance efficiency), subject to cycle time and precedence constraints. Because of the differences between models, individual tasks, task completion times and precedence constraints vary from model to model in the mixed-model problem. Consequently, two new assumptions are made. 1) Precedence constraints are consistent from model to model. That is, if task x precedes task y in any model there is no other model where task y must precede task x. 2) The same balance is used for all models. Tasks are not assigned to different stations for different models.

The mixed-model line balancing problem for a traditional straight line is reviewed in section 2. A model and solution algorithm for the mixed-model U-line balancing problem are presented in section 3. The algorithm is extended in the Appendix to the problem of balancing multiple mixed-model U-lines where multi-line stations with tasks from two or more U-lines may be formed. A discussion and summary follow in section 4.

2. Review of Mixed-Model Line Balancing for a Straight Line

Thomopolous [1967, 1970] proposed a four-step heuristic procedure for balancing a traditional straight line. We now review it with the aid of an illustrative example.

Example

Models A, B and C are produced, each at the rate of 25 units/planning period, on a production line running at a cycle time of 1.7 time units/model. Tasks, task processing times, and precedence constraints for each model are given in Table 1.

Step 1: Calculate the weighted average processing time for each task.

Total processing times for each task are calculated in Table 1. Consider, for example, task 1. It is completed 25 times for model A and 25 times for model C. Therefore, its total processing time is $25\times.5+25\times1=37.5$ time units. Total production during the planning period is 75 units, so the weighted average processing time is $37.5/75=.5$ time units/model.

Step 2: Merge the precedence graphs for all models into a single precedence graph.

Nodes and arcs on the merged precedence graph represent tasks and precedence constraints between tasks, respectively. If, for any model, task x is an immediate predecessor for task y, an arc xy is added to the merged precedence graph (if one does not already exist). The merged precedence graph for the example is shown in Figure 1.

Step 3: Solve a single-model line balancing problem with the tasks and average processing times from step 1 and the merged precedence graph from step 2.

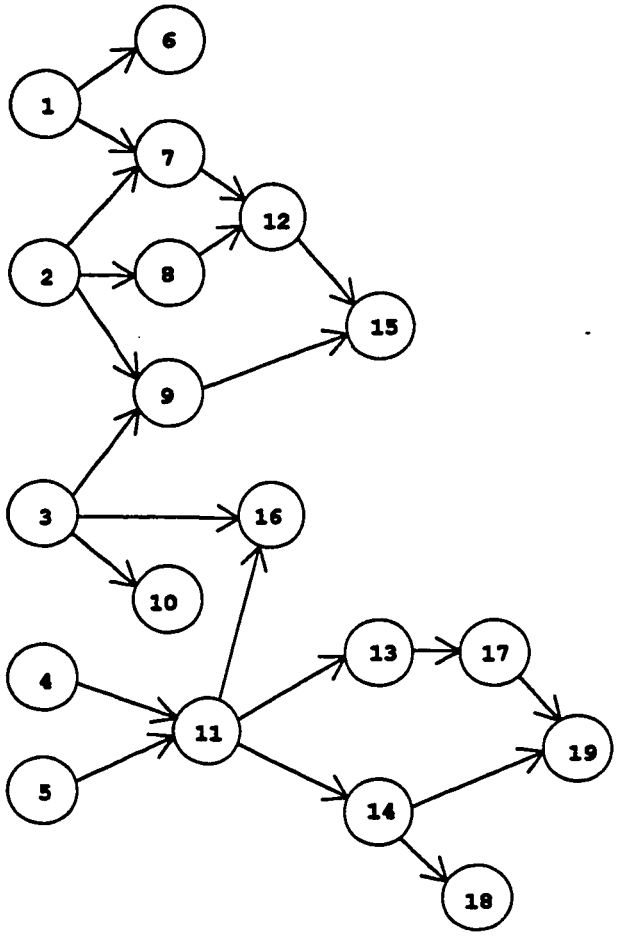
Table 1.

Task Data for Example in Section 2

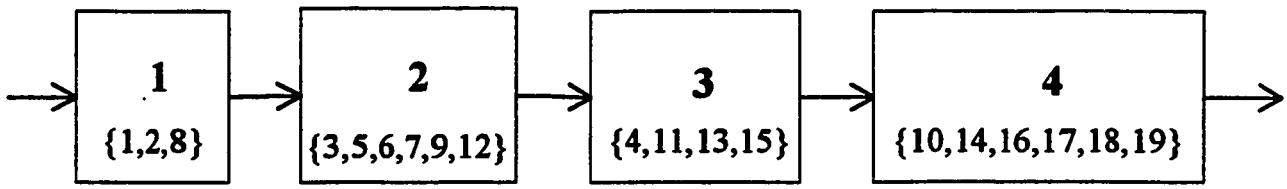
Task	Precedence Constraints			Processing Times					Distance
	A	B	C	A	B	C	Total	Average	
1				.5	0	1.0	37.5	.5	6
2				.4	.8	1.2	60	.8	7
3				0	.2	.4	15	.2	4
4				.4	0	0	10	.133	4
5				.2	.2	.2	15	.2	8
6	1			.2	0	0	5	.067	2
7	1,2	2	1,2	.4	.5	.6	37.5	.5	8
8		2	2	0	.5	.5	25	.333	9
9	2	2,3	2,3	.4	.3	.2	22.5	.3	8
10			3	0	0	.2	5	.067	2
11	4,5	5	5	.3	.3	.3	22.5	.3	5
12	7	7,8	7,8	.1	.3	.5	22.5	.3	9
13	11		11	.1	0	.1	5	.067	9
14	11	11	11	.2	.2	.2	15	.2	5
15	9,12	9,12	9,12	.7	1.0	1.5	80	1.067	10
16		3		0	.1	0	2.5	.033	9
17	13	11		.5	.5	0	25	.333	2
18	14	14	14	.3	.5	.3	27.5	.367	10
19	14,17	14,17		.4	.3	0	17.5	.233	9

Figure 1

1. Merged Precedence Graph for Example in Section 2



2. Straight Line Balance for Example in Section 2



Any single-model line balancing algorithm can be used to solve this problem. The solution for the example is shown in Figure 1. Although it is optimal for the average task processing times from step 1 and merged precedence graph from step 2, it may not even be feasible for individual models. Table 2 shows the times each model requires in each station. In three cases the required time exceeds the available time. This variation in station times is called *model imbalance* and can lead to production disruptions.

Step 4: Smooth the balance from step 3 to reduce model imbalance.

The balance determined in step 3 is adjusted by exchanging tasks between adjacent stations in order to reduce model imbalance. On a straight line, the objective for smoothing is to balance the line perfectly for each individual model. In this example, tasks 9 and 12 from station 2 are exchanged with tasks task 4 from station 3, task 15 from station 3 is exchanged with tasks 10, 14, 16 and 17 from station 4, and task 1 from station 1 is moved to station 2. This reduces the model imbalance as shown in part 2 of Table 2. Although there are still three cases when the required time exceeds the available time, the differences are smaller.

Model sequencing can now be applied to the final balance from step 4 to lessen the effect of any remaining imbalance. Model sequencing (see problem 2 in the previous section) selects the sequence in which models will be produced on the line.

Table 2.

Total Time Required by each Model

1. Initial Balance

Model	Station 1 {1,2,8}	Station 2 {3,5,6,7,9,12}	Station 3 {4,11,13,15}	Station 4 {10,14,16,17,18,19}
A	0.9	1.3	1.5	1.4
B	1.3	1.5	1.3	1.6
C	2.7	1.9	1.9	0.7
Average	1.63	1.57	1.57	1.23

Note: xx denotes an instance when the total time required by a model exceeds the cycle time of 1.7.

2. Final Balance After Smoothing

Model	Station 1 {2,8}	Station 2 {1,3,4,5,6,7}	Station 3 {9-14,16,17}	Station 4 {15,18,19}
A	0.4	1.7	1.6	1.4
B	1.3	0.9	1.7	1.8
C	1.7	2.2	1.5	1.8
Average	1.13	1.60	1.60	1.67

3. The Mixed-Model U-Line Balancing (MMULB) Problem

3.1 Notation

Differences between straight lines and U-lines are greater in mixed-model production than in single-model production. In a station on a straight line, all tasks performed by the operator during a cycle are on a single unit and thus a single model. In a station on a U-line the tasks performed on the front of the U-line are for a different unit, and frequently a different model, than those on the back of the line. The mixing of models in a single station allows the establishment of a single objective for smoothing on U-lines. This objective will remain in effect over the entire production sequence.

To illustrate, consider the U-line in part 1 of Figure 2. The operator in station 1 completes tasks 1 and 3 on model A on the front of the U-line, and then crosses to the back to complete tasks 12 and 11 on model C. Suppose the model sequence is CDACBA. Part 2 of the figure shows the many different model mixes that will be produced at stations 1 and 2.

Define the following notation.

Indices

x — task

n — model

m — station

Parameters

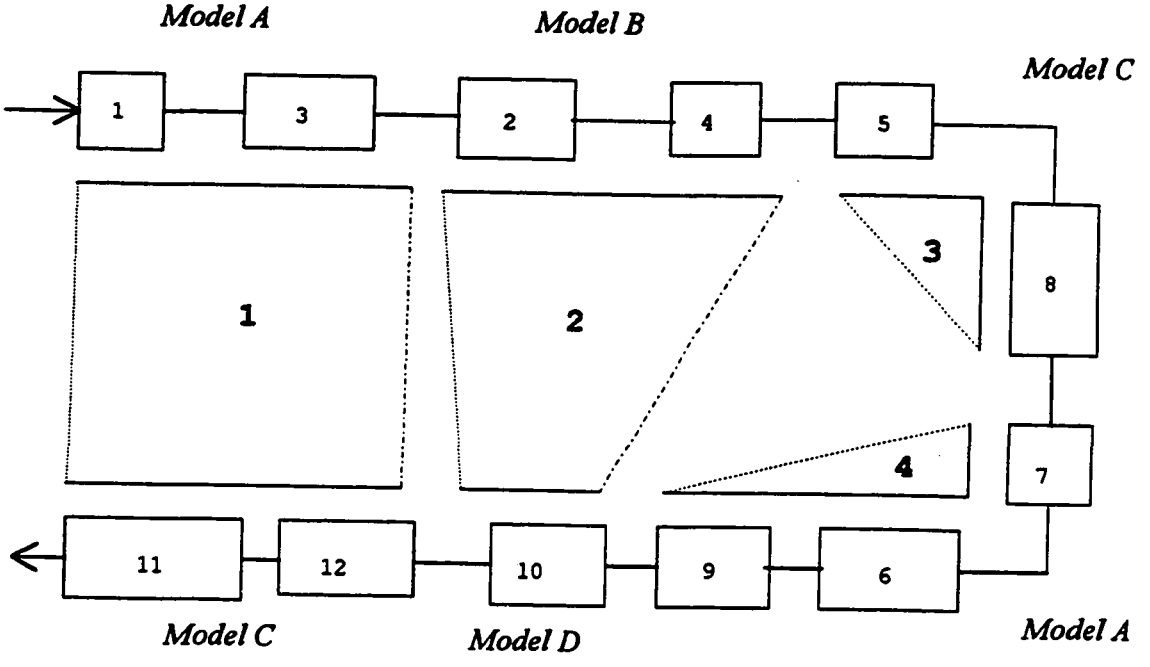
N — number of models

S — set of tasks for all models

X — total number of tasks

Figure 2

1. Mixed-Model Production on a U-line



2. Model Mixes in Stations 1 and 2

	Station 1		Station 2	
	front {1,3}	back {11,12}	front {2,4}	back {10}
Models produced				
cycle 1	A	C	B	D
cycle 2	C	D	A	A
cycle 3	D	A	C	C
cycle 4	A	C	D	B
cycle 5	C	B	A	A
cycle 6	B	A	C	C

P — set of merged precedence constraints for all models

$t_{n,x}$ — task processing time for task x on model n

$\ell_{n,x}$ — travel distance for task x on model n

g — time required to travel one unit of travel distance

d_n — number of units of model n produced during the planning period

Calculated Parameters

$\overline{t}_x = (\sum^{N}_{n=1} t_{n,x}d_n)/(\sum^{N}_{n=1} d_n)$ — weighted average processing time for task x

$\overline{\ell}_x = (\sum^{N}_{n=1} \ell_{n,x}d_n)/(\sum^{N}_{n=1} d_n)$ — weighted average travel distance for task x

$C = (\sum^{X}_{x=1} \overline{t}_x)/(\sum^{N}_{n=1} d_n)$ — cycle time

Decision Variables

M — total number of stations in the line balance

ST^F_m — tasks in station m located on the front of the U-line

ST^B_m — tasks in station m located on the back of the U-line

Calculated Decision Variables

$ST_m = ST^F_m \cup ST^B_m$ — set of tasks assigned to station m

DT_m — total travel distance in station m

$[f,b]_m$ — a model mix at station m: i.e., models f and b are produced at the front and back, respectively, of station m

$d([f,b]_m)$ — number of units of model mix $[f,b]_m$ produced during the planning period at station m

3.2 A Model for the MMULB Problem

Minimize M

Subject to:

$$\bigcup_{m=1}^{M} ST_m = S \qquad (1)$$

$$ST_m \cap_{m \neq m'} ST_{m'} = \varnothing \qquad \text{for all m} \qquad (2)$$

$$\sum_{x \in ST_m} \overline{t}_x + g \times DT_m \leq C \qquad \text{for all m} \qquad (3)$$

For every task $y \in S$

Either: If $(x,y) \in P$, $x \in ST_m$, $y \in ST_{m'}$, then $m \leq m'$, for all x; (4)

Or: If $(y,z) \in P$, $y \in ST_{m'}$, $z \in ST_{m''}$, then $m'' \leq m'$, for all z.

Location constraints (5)

The model is similar to the single U-line balancing with travel (SULB-T) problem with two differences:

1. Constraint (3) ensures that the sum of the weighted average task processing times for each model and the travel time in each station, does not exceed the cycle time.
2. P, the merged precedence graph is used in constraint (4).

3.3 A Solution Algorithm for the MMULB Problem

Step 1: Calculate the weighted average processing time and weighted average travel distance for each task. Calculate the cycle time (if required).

Step 2: Merge each model's precedence graph into a single precedence graph.

Step 3: Determine a balance.

Use the SULB-T branch and bound algorithm to find the optimal solution for the problem parameters determined in steps 1 and 2. Call the solution the initial balance.

Step 4: Smooth the initial balance from step 3 to reduce model imbalance.

Use the *smoothing algorithm* developed in the next section (section 3.4) to smooth the initial balance determined in step 3. Call the result the final balance.

3.4 A Smoothing Algorithm for the MMULB Problem

A smoothing algorithm adjusts the initial balance to reduce the level of model imbalance based on a given model sequence. This gives the final balance. At that point, model sequencing may be used to lessen the effect of any remaining imbalance. To improve the balance, different model sequences may be tested and the line rebalanced. Model imbalance is measured by comparing the times required by models at stations with target times. Usually the target is the cycle time (see measures $ib^{(1)}$ and $ib^{(2)}$ below) but other targets may also be used (see measure $ib^{(3)}$ below). On a U-line because tasks from different models are mixed in U-line stations a single goal, G, may be defined for every station, for every cycle. Define the following additional notation.

$$TT([f,b]_m) = \sum_{x \in ST_m} t_{f,x} + \sum_{x \in ST_m} t_{b,x} + g \times DT_m$$

— total time in station m for model mix $[f,b]_m$

$$G = \sum_{m=1}^{M} \{ \sum_{\text{all } [f,b]_m \text{ in } m} TT([f,b]_m) d([f,b]_m) \; / \; \sum_{\text{all } [f,b]_m \text{ in } m} d([f,b]_m) \} / M$$

— target for total time in a station

$$a^{(1)}([f,b]_m) = \begin{cases} 1 & \text{if } TT([f,b]_m) > C \\ 0 & \text{otherwise} \end{cases}$$

— variable indicating whether total time exceeds cycle time at station m for model mix $[f,b]_m$

$$a^{(2)}([f,b]_m) = \max\{0, TT([f,b]_m) - C\}$$

— amount, if any, that total time exceeds cycle time station m for model mix $[f,b]_m$

$$a^{(3)}([f,b]_m) = |TT([f,b]_m) - G|$$

— absolute deviation of total time from goal in station m for model mix $[f,b]_m$

$$ib^i_m = \sum_{\text{all } [f,b]_m \text{ in } m} d([f,b]_m) \times a^i([f,b]_m)$$

— imbalance measure i for station m. i = 1, 2 or 3.

$IB^i = \sum^M_{m=1} ib^i_m$ — imbalance measure i for the line. i = 1, 2 or 3.

The smoothing procedure exchanges tasks between stations so that the value of the imbalance measure IB^i decreases. The following observations suggest a procedure for selecting the tasks that should be exchanged.

1. Tasks with high processing time variances are more likely to cause model imbalance than tasks with low variances.
2. Pairs of tasks with positively-correlated processing times are more likely to cause model imbalance than pairs of tasks with negatively-correlated processing times.
3. A task whose processing time is positively-correlated with the total time in a station is more likely to cause model imbalance than a task whose processing time is negatively correlated with total time in the station.

An implication of these observations is the following. The level of imbalance in station m may be lowered by exchanging a task, whose processing time has a high variance and a high positive-correlation with total time in the station, with a task from another station m′ whose processing time has a low variance and a negative-correlation with total time in station m. Define the following additional notation.

$\bar{t}_x = [\sum^N_{n=1} t_{n,x}]/N$ — average processing time for task x

$T_{n,m} = \sum_{x \in ST_m} t_{n,x}$ — total time for model n in station m

$\bar{T}_m = [\sum^N_{n=1} T_{n,m}]/N$ — average total time in station m

$\sigma^2_x = [\sum^N_{n=1} (t_{n,x} - \bar{t}_x)^2]/(N-1)$ — variance of processing time for task x

$\sigma^2_m = [\sum^N_{n=1} (T_{n,m} - \bar{T}_m)^2]/(N-1)$ — variance of total time in station m

$\rho_{x,y} = [\sum^{N}_{n=1} (t_{n,x} - \bar{t}_x)(t_{n,y} - \bar{t}_y)]/\sigma_x\sigma_y$

— correlation coefficient of processing times for task x and task y

$\rho_{x,m} = [\sum^{N}_{n=1} (t_{n,x} - \bar{t}_x)(T_{n,m} - \bar{T}_m)]/\sigma_x\sigma_m$

— correlation coefficient of processing time for task x and total time in station m

Smoothing Algorithm:

1. Select a measure of model imbalance. (If necessary calculate G_m for each station m to reflect travel differences.)
   Specify K the number of exchanges to be considered.
   Calculate the imbalance ib_m for each station m. Calculate IB and denote it IB_0.
   Calculate σ^2_x and $\rho_{x,y}$, for all tasks x,y.
   For each station, sort tasks in decreasing order of σ^2_x.
2. Select the (next) station m. (This is the one with the largest imbalance.)
   Exit algorithm when none is left.
3. Select the (next) task x. (This is the one with the largest σ^2_x). If none is left go to 4.
   Calculate $\rho_{x,m}$ (correlation coefficient between processing time for x and total time in m).
   If $\rho_{x,m} \leq 0$ go to 3.
   Set $Y' = \varnothing$.
   For each task x′ in station m′=m-1, m+1
   If $\rho_{x,x'} < 0$ **and** exchanging x, x′ does not violate precedence or location constraints **then**
   If exchanging x, x′ violates the cycle time constraints **then**
   Check remaining tasks in m′ to see whether one or more can be included with x′ in the exchange (i.e., such tasks will violate precedence or cycle time constraints). Place these tasks in Y′. If $Y' = \varnothing$ go to 3.
   Calculate IB when x and $x' \cup Y'$ are exchanged. Denote this $IB(x, x' \cup Y')$.
   Increment k→k+1. If k>K go to 4.
   Go to 3.
4. Select the exchange $x, x' \cup Y'$ with the largest $IB_0 - IB(x, x' \cup Y') > 0$. Make the exchange.
   Calculate ib_m for each station m. Set $IB_0 \rightarrow IB(x, x' \cup Y')$.
   Go to 2.

Example

Suppose the three models from the example in section 2 are to be produced on a U-line with the model sequence ABCABC.... Suppose also that model imbalance measure $IB^{(3)}$ is used. Then the MMULB solution algorithm proceeds as follows.

Step 1: Calculate weighted average processing time and weighted average travel distance for each task. Calculate the cycle time (if required).

This was done in Table 1. Cycle time is specified; C=1.7.

Step 2: Merge each model's precedence graph into a single precedence graph.

This was done in Figure 1.

Step 3: Determine a balance.

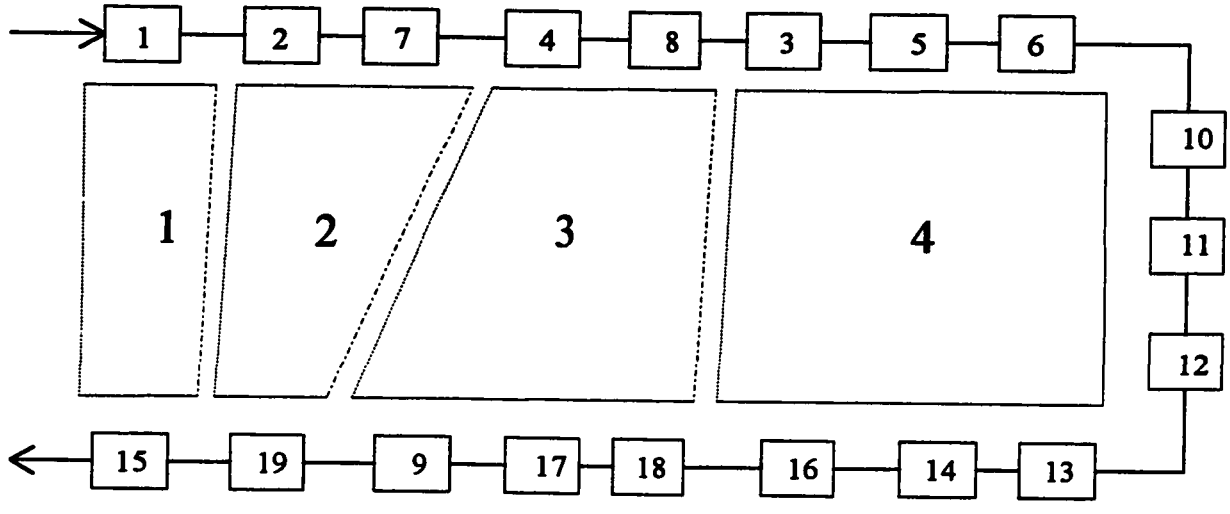
The SULB-T branch and bound algorithm from Part II of the thesis produced the optimal solution shown in Figure 3 for the problem parameters specified in steps 1 and 2.

Step 4: Smooth the balance from step 3 to reduce model imbalance.

- The target for the total time in a station is calculated from the initial balance in Figure 3; G = (1.659+1.682+1.653+1.635)/4 = 1.657.
- Values of model imbalance measure $ib^{(3)}_m$ are calculated in part 1 of Table 3 for each station m=1,2,3,4 in the intial balance from step 3.
- Next, the task processing time variances and correlation coefficients are calculated (see Table 4).
- Station 1 has the largest imbalance and so efforts start here to reduce imbalance by exchanging tasks between stations.

Figure 3

Initial Balance for Example in Section 3



Station:	1	2	3	4
Weighted average task processing time:	1.567	1.533	1.466	1.434
Weighted average travel time:	0.092	0.149	0.187	0.201
Weighted average total time:	1.659	1.682	1.653	1.635

Station time goal, G = (1.659+1.682+1.653+1.635)/4 = 1.657

Table 3

Calculations for MMULB Example in Section 3

1. Initial U-line balance

Station, m Tasks, ST^F_m, ST^B_m	1 {1}, {15}			2 {2,7}, {19}			3 {4,8}, {9,17,18}			4 {3,5,6,10,11,12,13,14,16}, ∅		
Model mix $[f,b]_m$	A,A	B,B	C,C	A,C	B,A	C,B	A,B	A,C	C,A	A	B	C
Total time, $TT([f,b]_m)$	1.2	1	2.3	1.1	1.7	2.1	1.7	1	1.7	1.1	1.3	1.9
Difference, $a^{(3)}([f,b]_m)$	.365	.565	.935	.408	.192	.592	.23	.47	.23	.356	.156	.444
Imbalance, $ib^{(3)}_m$	1.87			1.19			.93			.96		
$IB^{(3)}$	4.94											

2. Exchanges to reduce model imbalance

	Exchange		Imbalance	
	Staton, {tasks}	Station, {tasks}	$\{ib^{(3)}_1, ib^{(3)}_2, ib^{(3)}_3, ib^{(3)}_4\}$	$IB^{(3)}$
Initial balance	—	—	{1.87, 1.19, .93, .96}	4.94
Iteration 1	1, {15}	2, {2,19}	{1.37, .80, .93, .96}	4.04
Iteration 2	4, {12}	3, {17}	{1.37, .80, .27, .34}	2.77
Iteration 3	2, {7}	3, {4,8}	{1.37, .71, .17, .34}	2.59

3. Final U-line balance

Station, m Tasks, ST^F_m, ST^B_m	1 {1,2}, {19}			2 {4,8}, {15}			3 {7,12}, {9,18}			4 {3,5,6,10,11,13,14,16,17}, ∅		
Model mix $[f,b]_m$	A,A	B,B	C,C	A,C	B,A	C,B	A,B	A,C	C,A	A	B	C
Total time, $TT([f,b]_m)$	1.3	1.1	2.2	1.9	1.2	1.5	1.4	1.5	1.4	1.5	1.5	1.2
Difference, $a^{(3)}([f,b]_m)$	.265	.465	.635	.392	.308	.008	.07	.03	.07	.044	.044	.256
Imbalance, $ib^{(3)}_m$	1.37			.08			.17			.34		
$IB^{(3)}$	2.59											

Table 4

Task Processing Time Variance σ^2_x and Correlation Coefficients ρ_{xy} for Example in Section 3

	$\rho_{x,y}$																		σ^2_x
y x	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
1	.6	.6	-.1	0	-.1	.6	.1	-.6	.9.	0	.6	.8	0	.7	-.8	-.9	-.8	-.8	.253
2		1	-.9	0	-.9	1	.9	-1	.9	0	1	0	0	1	0	-.9	0	-1	.160
3			-.9	0	1	.9	-1	.9	.9	0	1	0	0	1	0	-.9	0	-1	.040
4				0	1	-.9	-1	.9	-.5	0	-.5	-.9	0	-.8	-.5	.5	-.5	.7	.053
5					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6						-.9	-1	.9	-.5	0	-.5	-.9	0	-.8	-.5	.5	-.5	.7	.013
7							.9	-1	.9	0	.9	1	0	1	0	-.9	0	-1	.250
8								-.9	.5	0	.9	-.5	0	.8	.5	-.5	.5	-.7	.111
9									-.9	0	-.9	-1	0	-1	0	.9	0	1	.010
10										0	.9	.5	0	.9	-.5	-1	-.5	-1	.013
11											0	0	0	0	0	0	0	0	0
12												0	0	1	0	-.9	0	-1	.040
13													0	.1	-1	-.5	-1	-.3	.003
14														0	0	0	0	0	0
15															-.1	-.9	-.1	-1	.163
16																.5	1	.3	.003
17																	.5	1	.111
18																		.3	.013
19																			.043

- Task 1 has the largest variance in station 1 and so is considered first. The correlation coefficient between task 1's processing time and total time in station 1 is calculated next, $\rho_{x=1,m=1} = .92$. Since $\rho > 0$ we attempt to exchange it with a task in station 2.
- Of the three tasks in station 2, $x' = 2,7,19$, only $x' = 19$ has $\rho_{x=1,x'=19} = -.8 < 0$. However exchanging tasks 1 and 19 violates the precedence constraints.
- Next, task 15 in station 1 is considered. $\rho_{x=15,m=1} = .88 > 0$ and so we attempt to exchange task 15 with a task in station 2.
- Of the tasks in station 2, $x' = 2,7,19$, only $x' = 19$ has $\rho_{x=15,x'=19} = -1 < 0$. Exchanging task 15 with task 19 does not violate the precedence constraints. However it violates the cycle time constraint.
- The remaining tasks in station 2, $x' = 2,7$, are checked to see whether one or more can be included with task 19 in the exchange. Task 2 is checked first. Exchanging task 15 with tasks 19 and 2 does not violate the precedence or the cycle time constraints. So the exchange is feasible. The value of the imbalance measure is $IB^{(3)} = 4.04$ when this exchange is made. Next task 7 is checked. Exchanging task 15 with tasks 19 and 7 violates the precedence constraints.
- All tasks at station 1 have now been considered. The best (and only) exchange is task 15 from station 1 with tasks 2 and 19 from station 2. The exchange is made. (See iteration 1 in part 2 of Table 3).
- Now station 4 has the largest imbalance and so efforts to reduce imbalance by exchanging tasks between stations move to this station. (See iteration 2 in part 2 of Table 3).

- The smoothing algorithm continues in this way until all possible exchanges are made. The final balance is shown in part 3 of Table 3. Smoothing has reduced model imbalance from $IB^{(3)}=4.94$ to 2.59 with only two model mixes exceeding the cycle time. Note that if the same problem on a straight line achieved a perfect balance for each model there would still be four instances where cycle time was exceeded. The ability of U-lines to smooth mixed-model line balances beyond the level attainable on a straight line is clearly illustrated.

4. Discussion and Summary

Many manufacturers use mixed-model production to produce different products on the same line. This helps them provide their customers with a variety of products in a timely and cost effective manner. The mixed-model U-line balancing (MMULB) problem assigns the tasks required to produce all the products (or models) to a minimum number of stations on a U-shaped production line. U-lines are widely used in just-in-time production systems and are well-suited to mixed-model production.

It is well-known that the traditional assembly line balancing (ALB) problem is NP-hard. The ALB problem is a special case of the (single-model) U-line balancing problem, which, in turn, is a special case of the MMULB problem. So the MMULB problem is also NP-hard, and approximate solution algorithms are needed to solve instances of realistic size. A four-step algorithm is presented in this paper. It follows from an algorithm developed by Thomopoulos [1970] for the mixed-model ALB problem. The first two steps transform the multi-model problem into an "equivalent" single-model problem. The third

step finds the optimal balance to this problem (using a branch and bound algorithm from the literature). While the computational requirements for this algorithm are exponential, it is capable of solving single-model problem instances with up to 25 tasks. The fourth step adjusts the balance from the previous step to make it feasible for the original multi-model problem. The adjustment consists of interchanging tasks between adjacent stations. The computational requirements for this step are $O(X^3M)$.

This paper is the first study of the mixed-model U-line balancing problem. There are still many areas where more study is needed.

Algorithm for the "equivalent" single-model problem in step three

The branch and bound algorithm used in step three cannot consider model imbalance. If an algorithm that considers model imbalance could be developed, better initial balances could be obtained. This might produce better smoothing targets and enable the smoothing algorithm to produce better final balances.

Smoothing algorithm for step four

More research is needed to determine the effect of different measures of model imbalance on the performance of mixed-model U-lines, and the benefits of expanding exchanges to three or more stations. A problem with the smoothing algorithm presented in this paper is that it may become trapped at a local minimum. To prevent this the algorithm could be changed so that some exchanges that increase model imbalance are made. A statistical search technique such as simulated annealing could be used to do this.

Integrating line balancing with model sequencing

Line balancing (LB) and model sequencing (MS) together determine how effectively a mixed-model U-line works. Thus far, these problems have been considered sequentially. (For example, the solution to the LB problem becomes a parameter for the MS problem. The solution to the MS problem then becomes a parameter for the LB problem, and the process is repeated until the solutions converge.) While the LB and MS problems are each NP-hard, it may still be possible to develop good heuristics that solve them simultaneously and give better solutions than those produced by a sequential approach.

Multiple mixed-model U-lines

Some facilities contain numerous U-lines, some connected by multiline stations (Part III). Some of these U-lines may be mixed-model lines. Research on balancing U-lines in such facilities is needed. Some preliminary results are reported in the Appendix.

References

Bard, J.F., E. Dar-El, and A. Shtub, "An Analytic Framework for Sequencing Mixed-Model Assembly Lines," *Int. J. Prod. Res.*, 30, 35-48, (1992).

Baybars, I. "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem," *Mgmt. Sci.*, 32, 909-932 (1986).

Ghosh S. and R.J. Gagnon, "A Comprehensive Literature Review and Analysis of the Design, Balancing and Scheduling of Assembly Systems," *Int. J. Prod. Res.*, 27, 4, 637-670 (1989).

Hackman, S.T., M.J. Magazine and T. S. Wee, "Fast, Effective Algorithms For Simple Assembly Line Balancing Problems," *Oper. Res.*, 37, 6, 916-924 (1989).

Hall, R.W., *Zero Inventories*, Dow Jones-Irwin, Homewood, Illinois (1983).

Hoffmann, T.R., "Eureka: A Hybrid System For Assembly Line Balancing," *Mgmt. Sci.*, 38, 1, 39-47 (1992).

Johnson, R.V., "Optimally Balancing Large Assembly Lines with 'Fable' ", *Mgmt. Sci.*, 34, 1, 240-253 (1988).

Kubiak, W., "Minimizing Variation of Production Rates in Just-In-Time Systems: A Survey", *Eur. J. Oper. Res.*, 66, 259-271 (1993).

Macaskill, J.L.C., "Production Line Balances For Mixed-Model Lines," *Mgmt. Sci.*, 19, 4, 423-434 (1972).

Miltenburg, J. and T. Goldstein, "Developing Production Schedules that Balance Part Usage and Smooth Production Loads for Just-in-Time Production Systems", *Naval*

Research Logistics, 38, 893-910 (1991).

Miltenburg J. and J. Wijngaard, "The U-Line Balancing Problem." *Mgmt. Sci.*, 40, 10, 1378-1388 (1994).

Monden, Y., *Toyota Production System*, Second Edition, Industrial Engineering Press, Institute of Industrial Engineers, Norcross, GA, (1993).

Talbot, F.B., J. H. Patterson and W. V. Gehrlein, "A Comparative Evaluation of Heuristic Line Balancing Techniques," *Mgmt. Sci.*, 32, 4, 430-454 (1986).

Thomopoulos, N.T., "Line Balancing Sequencing For Miced Model Assembly," *Mgmt. Sci.*, 14,2, 59-75 (1967).

Thomopoulos, N.T., "Mixed Model Line Balancing With Smoothed Station Assignments," *Mgmt. Sci.*, 16, 9, 593-603 (1970).

Yano, C.A. and R. Rachamadugu, "Sequencing to Minimize Work Overload in Assembly Lines with Product Options," *Mgmt. Sci.*, 37, 572-586 (1991).

Appendix

The Mixed-Model Multiple U-Line Balancing (N-MMULB) Problem

Part III of this thesis formulates and solves the problem of balancing a facility with multiple U-lines where multiline stations with tasks from two or more U-lines may be formed. The problem is called the N U-line balancing problem (NULB). Let NM denote the number of mixed-model U-lines to be balanced simultaneously.

Furthermore assume:

1. The location of the U-lines is not fixed. Consequently tasks from every U-line may be included in a multiline station with tasks from every other U-line.
2. All U-lines operate at the same cycle time.
3. Access to the U-lines is restricted to the opening of each U. That is, multiline stations can only be located at the opening of each U. Stations are not permitted to cross each other. Consequently the number of multiline stations associated with any U-line cannot exceed two.
4. Travel distances between U-lines are known constants.

A Solution Algorithm for the N-MMULB Problem

The heuristic algorithm for the NULB (Part III) can be modified and used as follows to solve the N-MMULB problem.

Step 1: Calculate weighted average task times and distances for each U-line.

Calculate the weighted average task processing time and the weighted average travel distance for each task on each U-line.

Step 2: *Merge each model's precedence graph into a single precedence graph for each U-line.*

Step 3: *Balance each U-line separately.*

Balance each U-line separately using the SULB-T algorithm to find M^*_u, the minimum number of stations required to operate U-line u without multiline stations.

Step 4: *Enumerate potential multiline subsets.*

From each U-line u enumerate all subsets of tasks with the following two properties.

1) All tasks in the subset may be assigned to multiline stations.

2) When the subset of tasks is assigned to a multiline station, the remaining tasks can be assigned to $M^*_u - \delta$ stations, $\delta = 1$ or 2.

For each subset with these properties, sum the task processing time for each task in the subset. Sort subsets in increasing order of this sum. Denote the sorted subsets $S^{\delta}_{j(u)}$, $\delta = 1,2$ and $j = 1,2,3,....$

Step 5: *Assign the multiline subsets to multiline stations.*

Using the algorithm from Part III solve the *multiline station assignment problem.*

Step 6: *Smooth the balance from step 5.*

Use the *smoothing algorithm* developed in the next section to smooth the balance determined in step 5.

A Smoothing Algorithm for the N-MMULB Problem

An NULB-T balance involves both stations entirely on each of the NM U-lines and multiline stations. A unique goal, G_u, is set for each U-line, and G_A, for each multiline station m(A), where A =uv,uvw,... and u≠v≠w...:

$$G_u = [\sum_{m=1}^{M_u} ((\sum_{x \in ST_{m(u)}} \overline{t}_{x,u}) + g \times DT_{m(u)})]/ (M_u)$$

$$G_{(A)} = \sum_{x,u \in ST_{m(A)}} \overline{t}_{x,u} + g \times DT_{m(A)}, \quad A=uv,uvw,... \text{ and } u \neq v \neq w...$$

Smoothing Algorithm:

The algorithm first smooths each individual U-line and then smooths the multiline stations.

1. Set a goal for each U-line.
   Select a measure of model imbalance.
   Specify K and K_{multi}, the number of exchanges to be considered on each U-line and multiline station respectively.
   Calculate the imbalance $ib_{m(A)}$ for each station m(A). Calculate IB_u for the U-lines and $IB_{m(A)}$ for the multiline stations.
2. Select the U-line with the maximum IB_u and smooth the single line stations m(u) using the algorithm in section 3.4. Exchanges between the single line stations and any multi-line stations on u are allowed.
   Repeat 2 until all U-lines have been smoothed.
3. Calculate the goal for each multiline station m(A), A =uv,uvw,... and u≠v≠w...
   Calculate the imbalance $ib_{m(A)}$ for each multiline station m(A).
   Calculate $IB = \sum_{u=1 to NM} IB_u + \sum_{all\ m(A),\ A\ =\ uv,uvw...} IB_{m(A)}$. Set $IB_O = IB$.
4. Select the (next) station m(A). (The unsmoothed station with the largest imbalance.)
5. Select the (next) task x in m(A). (This is the one with the largest σ^2_x). If none is left go to 6. Calculate $\rho_{x,m}$ (correlation coefficient between processing time for x and total time in m(A)).

If $\rho_{x,m} \leq 0$ go to 5.

For each $u \in A$ analyse exchanges with stations adjacent to m(A).

Set $Y' = \varnothing$.

For each task x′ in stations m(u) adjacent to m(A)

If $\rho_{x,x'} < 0$ **and** exchanging x, x′ does not violate precedence or location constraints **then**

If exchanging x, x′ violates the cycle time constraints **then**

Check remaining tasks in m(u) to see whether one or more can be included with x′ in the exchange (i.e., such tasks will violate precedence or cycle time constraints). Place these tasks in Y′. If $Y' = \varnothing$ go to 5.

Recalculate IB when x and $x' \cup Y'$ are exchanged. Denote this $IB(x, x' \cup Y')$.

Increment $k \rightarrow k+1$. If $k > K_{multi}$ go to 6.

Go to 5.

6. **Select** the exchange $x, x' \cup Y'$ with the largest $IB_0 - IB(x, x' \cup Y') > 0$. Make the exchange.

Set $IB_0 \rightarrow IB(x, x' \cup Y')$.

Go to 4.

STOCHASTIC U-LINE BALANCING

Abstract

Recent assembly line balancing research has placed greater emphasis on problems with stochastic task processing times. Although some of the research involves just-in-time production systems, none has considered the problem of stochastic U-line balancing. The flexibility inherent in U-lines adds an additional level of complexity to the already difficult stochastic line balancing problem. Under certain assumptions dynamic programming may be used to balance a U-line. However, for more general problems finding an optimal solution is infeasible. For such problems, simulated annealing is utilized.

1. Introduction

A critical design and implementation issue for assembly line production is the assignment of production tasks to work stations, the assembly line balancing (ALB) problem. ALB formulations usually assume deterministic task processing times. Numerous algorithms exist for solving this NP-hard problem. In many production systems task times are not constant. Unfortunately, incorporating stochastic task times into assembly line balancing significantly complicates the problem. Not only must the number of stations be minimized, but tasks must be assigned to stations in a way which minimizes the disruptions caused by variations in total station times. Exceeding the cycle time in any station may cause line stoppages, defective production or may require extra labour to bring the station back to pace.

Stochastic Line Balancing with Normally Distributed Task Times

The first studies to incorporate stochastic task times into line balancing assumed normally distributed task times. Various strategies were employed to reduce the probability of stoppages due to station times exceeding the cycle time. Moodie and Young [1965] solved the problem using a largest task time heuristic and then attempted to transfer tasks between stations to equalize station time variance and minimize idle time. Reeve and Thomas [1973] began with an initial solution and attempted to rearrange the task elements in the stations so as to minimize the probability that one or more stations exceeds the cycle time.

Kao [1976] altered the cycle time constraint so that for each station the probability

that total station time is less than cycle time is greater than or equal to α. He then used a dynamic programming approach based on Mitten's [1974] preference order to balance the line. Kao's formulation and preference order did not necessarily result in optimal solutions (Sniedovich [1981]). Carraway [1989] revised Kao's algorithm to remove two sources of potential suboptimality, the effect of negative task times due to the assumption of normally distributed task times and the method proposed for resolving ties during the recursion. If the probability of any task having a negative task time is $\leq \alpha$ then the difficulty created by negative task times is eliminated. By relaxing the rules for resolving ties and retaining more subsets during the solution procedure the algorithm finds an optimal solution. Carraway introduced a second version of the algorithm with stronger preference measures. This version outperformed the first in instances where task time variances were relatively high.

Driscoll and Abel-Shafi [1985] used confidence levels to establish the task times to be used in line balancing. Each task duration was fixed according to the desired level of confidence and then the line was balanced using a ranking heuristic. This approach has the advantage of being able to use any deterministic solution algorithm but it lacks consistency. Any station with more than one task will exceed the level of confidence specified for individual tasks. The probability of station time exceeding cycle time will vary from station to station and the average level of confidence will be higher when cycle time is high and lower as cycle time decreases (and with it the average number of tasks per station). Their simulation model allowed the analysis of the effect of multiple factors including cycle time, line balance, line layout, model-mix and multiple operator stations.

Other constrained versions of the problem can also be formulated and solved as deterministic problems (Sphicas and Silverman [1976] and Sniedovich [1981]).

For general stochastic ALB formulations finding an optimal solution for most problems is not possible and heuristic techniques must be employed. Early efforts used Monte Carlo simulation, a capability incorporated into COMSOAL (Arcus [1966]). More recently, Nkasu and Leung [1995] used a statistical sampling approach based on COMSOAL to sample both cycle times and task times from specified distributions and randomly assign tasks to stations. The simulation allows sampling from a variety of distributions and the inclusion of single or multiple objectives.

Another statistical search technique which has proven effective for stochastic ALB is simulated annealing (Suresh and Sahu [1994]). Simulated annealing (SA) is based on an analogy between a minimization optimization problem and the physical process of finding low energy states of a solid by slowly cooling a melted metal. If cooling is too rapid the metal will solidify too quickly and will not attain the lowest energy state. Similarly, in optimization, moving too quickly to a solution may lead to a local minimum. Numerous papers describe the SA process including Kirkpatrick et al. [1983], Glover and Greenburg [1989], and Eglese [1989]. Schmidt and Jackman [1995] use SA in a three tier multi-echelon approach to assembly line design aimed at minimizing the cost of operating prototype automated assembly lines.

Balancing U-lines

The main difference between balancing U-lines and straight lines is the ability to create *crossover stations*. These crossover stations are responsible for many of the

performance differences between the two layouts. Operating differences examined in parts I to IV in this thesis relate to the construction and functioning of individual stations. When task processing times are stochastic the interaction between stations becomes important, since a problem in one station can affect production in succeeding stations. These interactions differ between straight lines and U-lines. A straight line station, m, interacts only with the preceding and succeeding stations. A problem in station m-1 may delay station m which in turn affects station m+1. This is also the case for non-crossover U-line stations, but a crossover station may interact with as many as 4 different stations, two preceding and two following. In Figure 1, exceeding cycle time in stations 1 or 4 may affect station 2 which in turn may delay its successor stations 1 and 3. The nature of production on the U-line determines the impact of exceeding cycle time.

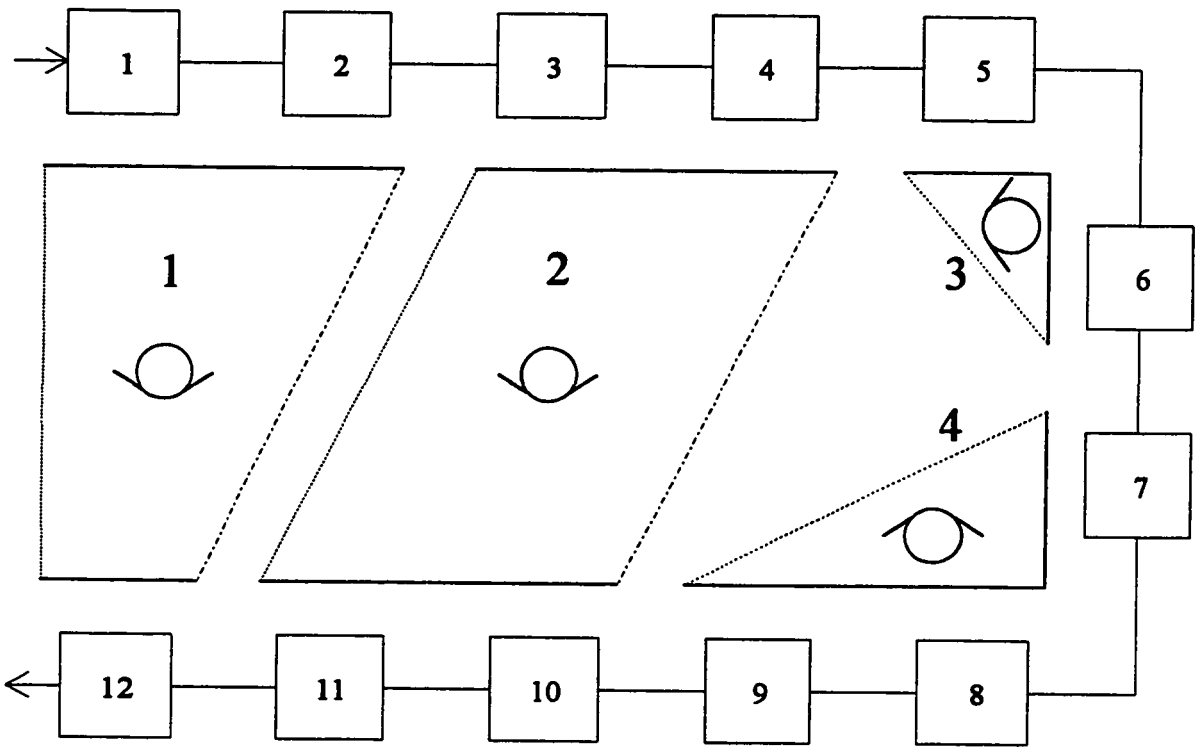
The stochastic U-line balancing (STULB) problem is examined in this paper. The study begins with a model and DP algorithm for solving the U-line equivalent to Carraway's problem (2). In section 3 a simulated annealing algorithm is offered for solving general STULB problems heuristically. Deliberate unbalancing of an assembly line to increase throughput is discussed in section 4. Discussion follows.

2. A Dynamic Programming Algorithm for Solving the Stochastic U-Line Balancing Problem

If the STULB problem includes the constraint that the probability that total station time is less than or equal to the cycle time must be at least α, then it may be solved using Carraway's DP algorithm.

Figure 1

A Typical U-line Balance



Legend

Task distance

Return distance

Crossover distance

2.1 Notation

The following notation will be used in this paper.

S = the set of tasks to be assigned.

X = the number of tasks in S.

P = the set of precedence relations for S.

C = the cycle time.

$\mathcal{S}$= the complete set of feasible subsets of S relative to P.

t_x = task time for task x. Task times are assumed to be independently and normally distributed random variables with mean μ_x and variance, σ^2_x.

$\mathcal{F}_x$ = the distribution function for task time for task x.

ST_m = the set of tasks assigned to station m.

$\tau_m = \sum_{x \in STm} t_i$, the work content at station m.

$\alpha = Pr(\tau_m \leq C)$, the minimum probability that total station time will be $\leq C$.

M = the number of stations required to balance the line.

S_i = a feasible subset of tasks of S.

$F[S_i]$ = the minimum cost of feasible subset S_i. Cost is defined as the number of full stations required to complete the tasks in S_i plus the time in the last station.

2.2 A Model For The Stochastic U-Line Balancing Problem

The stochastic U-line balancing problem may be modelled as:

Minimize M

Subject to:

$$\bigcup_{m=1}^{M} ST_m = S \qquad (1)$$

$$ST_u \bigcap_{u \neq v} ST_v = \varnothing \qquad (2)$$

$$P(\tau_m \leq C) \geq \alpha \qquad \text{for } m = 1,\ldots,M \qquad (3)$$

For every task $y \in S$ (4)

Either: If $(x,y) \in P$, $x \in ST_m$, $y \in ST_{m'}$, then $m \leq m'$, for all x;

Or: If $(y,z) \in P$, $y \in ST_m$, $z \in ST_{m'}$, then $m' \leq m$, for all z.

Task times are assumed to be normally distributed. This model differs from other ULB problems in constraint (3) where the probability that total time in any station is less than or equal to cycle time must be at least α.

2.3 A Dynamic Programming Algorithm For Stochastic U-Line Balancing

An optimal solution may be found for the problem defined in 2.2 by modifying the *preference order dynamic programming* (DP) algorithm of Carraway [1989].

The Cost Function

If a subset of tasks, S_i, is assigned to m stations, the cost function, ∇, for adding task j to the end of S_i is defined as:

$$\nabla[m,N(\mu,\sigma^2),j] = \begin{cases} (m,\ N(\mu+\mu_j,\sigma^2+\sigma_j^2)) & \text{if } [N(\mu+\mu_j,\sigma^2+\sigma_j^2)]^{-1}(\alpha) \leq C \\ (m+1,\ N(\mu_j,\sigma_j^2)) & \text{otherwise} \end{cases}$$

S_i is assigned to m stations, with time in the last station distributed $N(\mu,\sigma^2)$. Adding task

j to the end of this subset will result in the time in the last station, τ_m, being distributed $N(\mu+\mu_j,\sigma^2+\sigma_j^2)$. If $\mathcal{F}$ is the set of all normal distribution functions with non-negative means and $f\in\mathcal{F}$, Carraway defines $f^{-1}(\alpha) = \inf\{t|\ f(t) \geq \alpha\}$, the smallest time, t, such that the $Pr(x \leq t) \geq \alpha$. Thus if $P(\tau_m \leq C) \geq \alpha$ task j is assigned to station m, otherwise, the assignment of j to station m is not feasible with respect to (3) and j is assigned to a new station, m+1.

The Preference Order

Consider two ordered subsets which are arrangements of subset S_i. The subsets are assigned to stations with each subset requiring m_k stations, with time in the last station distributed $N(\mu_k,\sigma^2_k)$. Subset 1 with $(m_1,N(\mu_1,\sigma^2_1))$, is preferred to ($\succeq$) subset 2 with $(m_2, N(\mu_2,\sigma^2_2))$ if one of the following conditions exists.

Station dominance

$$m_1 < m_2. \tag{5}$$

Vector dominance

$$m_1 = m_2 \text{ and } [\,N(\mu_1+\mu,\sigma^2_1+\sigma^2)]^{-1}(\alpha) \leq [\,N(\mu_2+\mu,\sigma^2_2+\sigma^2)]^{-1}(\alpha). \tag{6}$$

A preference relation exists if the number of stations for subset 1 is less than for subset 2, or if the number of stations is equal and the α fractile for the distribution created by adding any μ and σ^2 to the subset 1 is no greater than the α fractile for the corresponding subset 2 distribution.

Condition (6) may be restated as

$$\mu_1 + \mu + Z_\alpha(\sigma^2_1+\sigma^2)^{1/2} \leq \mu_2+\mu + Z_\alpha(\sigma^2_2+\sigma^2)^{1/2}. \tag{7}$$

Condition (7) holds under the following circumstances (Carraway, [1989])

a) if $\alpha > 0.5$, $\mu_1 \leq \mu_2$ and $\sigma^2_1 \leq \sigma^2_2$. (8)

b) if $\alpha < 0.5$, $\mu_1 \leq \mu_2$ and $\sigma^2_1 \geq \sigma^2_2$. (9)

c) if $\alpha > 0.5$, $\mu_1 \leq \mu_2$ and $\sigma^2_1 > \sigma^2_2$ then (7) also holds if

$$Z_\alpha \leq (\mu_2 - \mu_1)/(\sigma_1 - \sigma_2). \quad (10)$$

d) if $\alpha < 0.5$, $\mu_1 \leq \mu_2$ and $\sigma^2_1 > \sigma^2_2$ then (7) also holds if

$$Z_\alpha \geq (\mu_2 - \mu_1)/(\sigma_1 - \sigma_2). \quad (11)$$

If any of conditions (5) - (11) apply then subset 1 is preferred to subset 2 and subset 2 may be eliminated from further consideration. If not, both subsets are retained for further consideration.

Dynamic Programming Recursion

Carraway describes the set of subsets which are not dominated as a set of *maximal elements* of *S*. A set of *maximal elements* of a set Y with respect to the preference relation $\succeq$ is defined as:

$$\text{maxl}(Y, \succeq) = \{y \in Y \mid \nexists\, y' \in Y \text{ such that } y' \succeq y \text{ and } y \succ y'\}.$$

An element y is in maxl(Y, $\succeq$) if there is no y′ in Y that is preferred to Y. The dynamic programming recursion based on the preference ordering is:

$$F(\varnothing) = \{[0, N(0,0)]\},$$

$$F(S_i) = \underset{S_i\text{-}j \in S}{\text{maxl}} \{\cup (\nabla[m,N(\mu,\sigma^2),j] \mid (m,N(\mu,\sigma^2) \in F(S_i\text{-}j)), \succeq\} \quad \text{for } S_i \in S$$

where *S* is the complete set of feasible subsets of S. For a given subset S_i the cost of all feasible arrangements S_i-j are compared and a preference order established. The

preference order will eliminate many S_i-j from further consideration leaving the set of maximal subsets with associated costs $F(S_i)$.

To avoid the problem created by the possibility of negative processing times either $\alpha > 0.5$ (Sniedovich[1981]) or $f_x(\alpha)^{-1} \geq 0$ for all $x \in S$, meaning that every task has at least a probability of α that processing time will be non-negative (Carraway [1989]). The use of this DP formulation for solving STULB problems is illustrated in the following example.

Example

Consider Bowman's [1960] problem with normally distributed task times, a cycle time of 22 and $\alpha = 0.95$. The precedence relationships, task times and variances are shown in Figure 2. Table 1 shows the first steps in the DP algorithm using a reaching approach to enumerate feasible subsets. Different orderings of subset S_i are denoted as $S_{i,k}$. While certain orderings of each S_i may be eliminated through the preference order, others must be retained for several steps. For example, after subsets of size 2 are enumerated the ordered subsets {1,7} and {7,1} are compared.

k	$S_{i,k}$	Last task, j	$f(S_{i,k})$
1	{1,7}	7	1+(10,3)
2	{7,1}	1	1+(11,1)

Station dominance: $m_1 = m_2 = 2$ so there is no station dominance.

Vector dominance: $\mu_1 < \mu_2$ so dominance exist if a) $\sigma^2_1 \leq \sigma^2_2$ or b) $Z_\alpha \leq (\mu_2-\mu_1)/(\sigma_1-\sigma_2)$.

a) $\sigma^2_1 > \sigma^2_2$ so no dominance is established

b) $Z_{.95} = 1.645$ and $(\mu_2-\mu_1)/(\sigma_1-\sigma_2) = (11-10)/(1.732-1) = 1.366 < Z_{.95}$ so no

Figure 2

Precedence Relations for Example 1

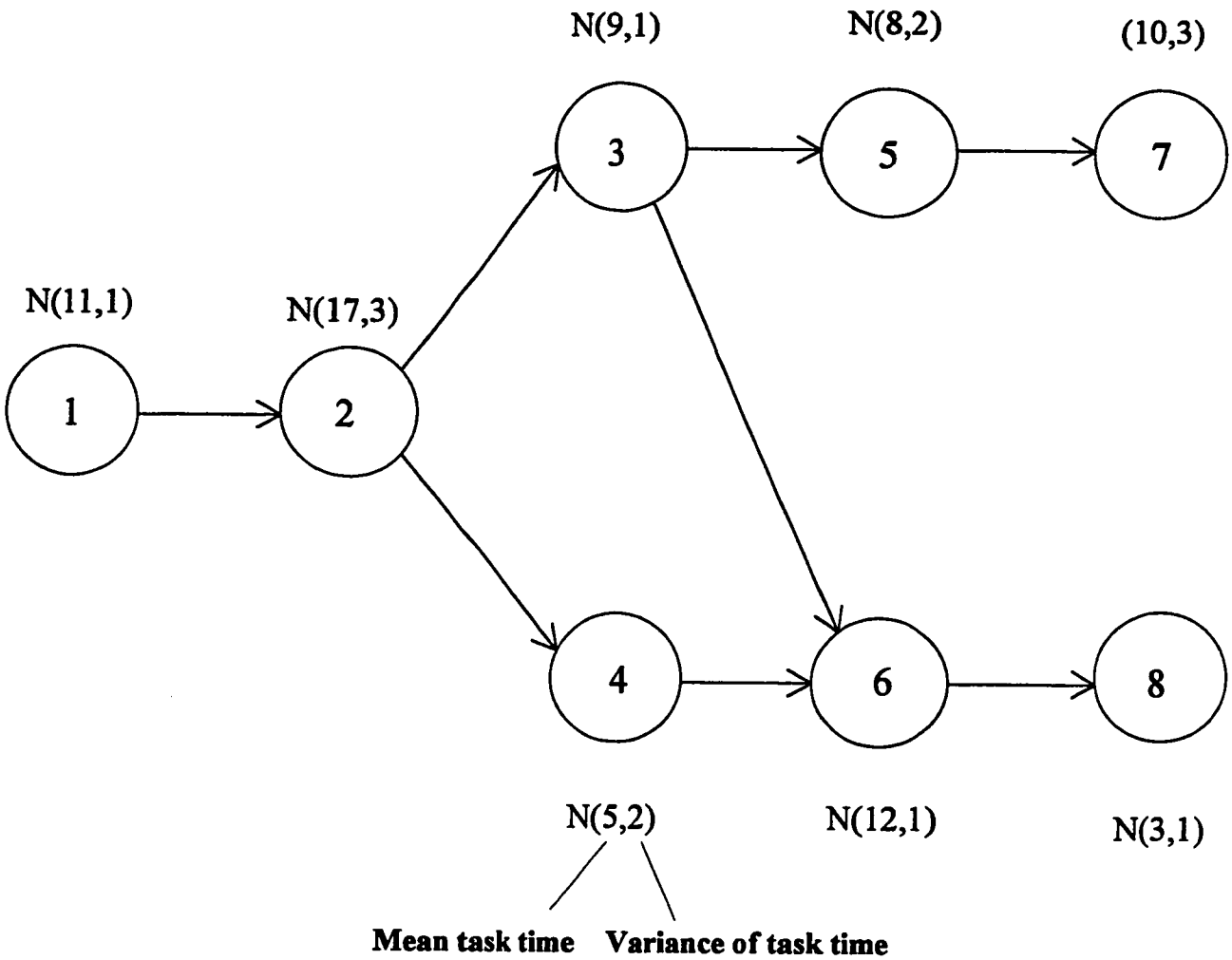


Table 1

An Illustrative Example of the Stochastic DP Algorithm

Size, n	i	$S_i^{(n-1)}$	j	$f[S_i^{(n-1)}]$		$S_i^{(n-1)}+\{1\}$	$S_i^{(n-1)}+\{2\}$	$S_i^{(n-1)}+\{3\}$	$S_i^{(n-1)}+\{4\}$	$S_i^{(n-1)}+\{5\}$	$S_i^{(n-1)}+\{6\}$	$S_i^{(n-1)}+\{7\}$	$S_i^{(n-1)}+\{8\}$
1	1	⌀	–	0	j $S_{1,j}^{(1)}$ $f[S_{1,j}^{(1)}]$	1 {1} 0+(11,1)						2 {7} 0+(10,3)	3 {8} 0+(3,1)
2	1	{1}	1	0+(11,1)	j $S_{1,j}^{(2)}$ $f[S_{1,j}^{(2)}]$		1 {1,2} 1+(17,3)					2 {1,7} 1+(10,3)	3 {1,8} 0+(14,2)
	2	{7}	7	0+(10,3)	j $S_{2,j}^{(2)}$ $f[S_{2,j}^{(2)}]$	1 {7,1} 1+(11,1)				2 {7,5} 0+(18,5)			3 {7,8} 0+(13,4)
	3	{8}	8	0+(3,1)	j $S_{3,j}^{(2)}$ $f[S_{3,j}^{(2)}]$	1 {8,1} 0+(14,2)					2 {8,6} 0+(15,2)		3 {8,7} 0+(13,4)
3	1	{1,2}	2	1+(17,3)	j $S_{1,j}^{(3)}$ $f[S_{1,j}^{(3)}]$			1 {1,2,3} 2+(9,1)	2 {1,2,4} 2+(5,2)			3 {1,2,7} 2+(10,3)	4 {1,2,8} 2+(3,1)
	2	{1,7}	7	1+(10,3)	j $S_{2,j}^{(3)}$ $f[S_{2,j}^{(3)}]$		1 {1,7,2} 2+(17,3)			2 {1,7,5} 1+(18,5)			3 {1,7,8} 1+(13,4)
	3	{7,1}	1	1+(11,1)	j $S_{3,j}^{(3)}$ $f[S_{3,j}^{(3)}]$		1 {7,1,2} 2+(17,3)			2 {7,1,5} 1+(19,3)			3 {7,1,8} 1+(14,2)
	4	{1,8}	8	0+(14,2)	j $S_{4,j}^{(3)}$ $f[S_{4,j}^{(3)}]$		1 {1,8,2} 1+(17,3)				2 {1,8,6} 1+(12,2)	3 {1,8,7} 1+(10,3)	
	5	{5,7}	5	0+(18,5)	j $S_{5,j}^{(3)}$ $f[S_{5,j}^{(3)}]$	1 {5,7,1} 1+(11,1)							2 {7,5,8} 1+(3,1)
	6	{6,8}	6	0+(15,2)	j $S_{6,j}^{(3)}$ $f[S_{6,j}^{(3)}]$	1 {6,8,1} 1+(11,1)			2 {6,8,4} 1+(5,2)			3 {6,8,7} 1+(10,3)	
	7	{7,8}	7	0+(13,4)	j $S_{7,j}^{(3)}$ $f[S_{7,j}^{(3)}]$	1 {7,8,1} 1+(11,1)				2 {7,8,5} 1+(8,2)	3 {7,8,6} 1+(12,1)		
etc.													

vector dominance is established. Both {1,7}, {7,1} are maximal elements of S and must be retained for the next step, enumerating subsets of size 3. Table 2 demonstrates the use of dominance rules on subsets of size 3.

2.4 Consideration Of Station Travel

One problem with Carraway's DP algorithm for U-line balancing is that there is no consideration of travel in the stations. As discussed in part II of this thesis, inclusion of station travel is an important factor in ensuring feasibility during U-line balancing. The DP algorithm can be modified to incorporate station travel but difficulty arises when attempting to establish preference relations. As tasks are assigned to stations the choice of task location affects the feasibility of later stations through their travel component. It may be impossible to establish a preference relationship for many subset pairs because the impact on travel in succeeding stations cannot be determined. If an optimal solution is desired, all arrangements of a subset which do not have the same beginning and ending points would have to be evaluated for their effect on the solution and in many cases no preference order would be established. Thus, finding an optimal solution will be computationally infeasible for larger problems.

One other travel consideration is the variability of operator travel times. Although some variability will exist, its impact will be minor compared to the task time variability. For this reason, travel time will be treated as a constant factor of travel distance with variance of 0.

Table 2

Dominance in Duplicate Subsets of Size n=3 in the Example

Subset	Subset Order	Last task	$f[S_i]$ $m+N(\mu, \sigma^2)$	Dominance Criteria	Retain
{1,2,3}	{1,2,3}	2	2+(9,1)		{1,2,3}
{1,2,4}	{1,2,4}	4	2+(5,2)		{1,2,4}
{1,2,7}	{1,2,7}	7	2+(10,3)		{1,2,7}
	{1,7,2}	2	2+(17,3)	vector(8)	
	{7,1,2}	2	2+(17,3)	vector(8)	
{1,2,8}	{1,8,2}	2	1+(17,3)		{1,8,2}
	{1,2,8}	8	2+(3,1)	station	
{1,5,7}	{7,5,1}	1	1+(11,1)		{7,5,1}
	{1,7,5}	5	1+(18,5)	vector(8)	
	{7,1,5}	5	1+(19,3)	vector(8)	
{1,7,8}	{7,8,1}	1	1+(11,1)		{7,8,1}
	{1,8,7}	7	1+(10,3)		{1,8,7}
	{1,7,8}	8	1+(13,4)	vector(8)	
	{7,1,8}	8	1+(14,2)	vector(8)	
{1,6,8}	{1,8,6}	6	1+(12,2)	vector(8)	
	{8,6,1}	1	1+(11,1)		{8,6,1}
{4,6,8}	{8,6,4}	4	1+(5,2)		{8,6,4}
{5,7,8}	{7,5,8}	8	1+(3,1)		{7,5,8}
	{7,8,5}	5	1+(8,2)	vector(8)	
{6,7,8}	{7,8,6}	6	1+(12,1)	vector(10)	
	{8,6,7}	7	1+(10,3)		{8,6,7}

3. A Simulated Annealing Approach To Stochastic U-line Balancing

To overcome the problems posed by operator travel and by the introduction of more general task time distributions a heuristic search technique is recommended. One method suited to U-line balancing is simulated annealing (SA). Because SA samples the solution space, problem structure may be more general and can easily accommodate travel and location constraints.

3.1 Notation

E — the balance measure for the U-line.

T — the simulated annealing temperature.

T_{in} — the initial simulated annealing temperature.

CR — the cooling ratio.

N_{it} — the number of iterations at one temperature.

icount — The algorithm will stop after icount cooling cycles without a change in the solution.

DT_m — the travel distance in station m.

$g \times DT_m$ — the time required to travel distance DT_m.

τ_m — the total time in station m. $\tau_m = \sum_{x \in ST_m} t_x + g \times DT_m$

3.2 Objectives for Simulated Annealing

The line balancing objective is to minimize some balance measure, E. Two objectives used on straight lines by Suresh and Sahu [1994] were based on early stochastic

line balancing research.

1. Minimize the probability of stopping the line, PS (Reeve [1971]).

$$E = PS = (1 - \prod_{m=1}^{M} PS_m)$$

where $PS_m = Pr\ (\tau_m > C)$.

2. Minimize the *smoothness index*, SI, a measure of station times relative to maximum station time (Moodie and Young [1965]).

$$E = SI = \sqrt{(\sum_{m=1 to M} (\tau_{max} - \tau_m)^2)}$$

Any other appropriate stochastic balance measure could used in the objective function. SA also has the ability to accommodate multiple criteria objectives.

3.3 A Simulated Annealing Algorithm for the STULB Problem

The following simulated annealing algorithm based on the work of Suresh and Sahu [1994] is proposed to solve the stochastic U-line balancing problem. An initial solution is found and then small changes are made to the current solution and the effect on the objective function value is assessed. All moves which improve E are accepted. If a move worsens E a probabilistic test is used to determine whether the move is accepted.

Step 1. Initialize the algorithm.

1.1 Find an initial solution.

A single pass heuristic which assigns tasks based on largest mean task time is find the initial solution.

1.2 Calculate the objective function value.

The objective function value, $E_{current}$, is calculated as the probability of stopping the

line, PS.

1.3 Set the SA parameters.

Initial values are selected for T_{in}, CR, N_{it} and icount. T_{in} is set using the procedure described in Lee and Iwata[1991] and Rose *et al.*[1990]. Its initial value is set relatively high to accept many different potential solutions.

Step 2. Generate a move to a neighbour solution.

A move to a neighbour solution is a small change in the current solution which results in a feasible solution. Two possibilities are allowed for generating neighbours, transfers and trades. One will be randomly selected.

Transfer a task to another station.

A task and station are randomly selected. If the task is not currently in the station it may be transferred into the station.

Trade two tasks between stations.

Two tasks are randomly selected. A neighbour may be generated by a trade where the two jobs are exchanged between their respective stations.

Unlike on a straight line where a task must be physically connected to its station, on a U-line a task may join a station through a crossover without moving. Thus two options exist for making a transfer or trade: 1) changing task locations or 2) changing operator paths. In some cases both will be feasible and the choice between the two is random.

All transfers or trades being considered must be feasible with respect to P and to location constraints or else the algorithm returns to the beginning of step 2.

Step 3. Determine whether to accept the new solution.

3.1 Calculate E_{move}, the objective function value for the new solution.

3.2 Compare E_{move} to $E_{current}$.

i) If $E_{move} \leq E_{current}$

The configuration is accepted and $E_{current} = E_{move}$. Compare the new value to the best solution found to this point, E_{best}. If $E_{current} < E_{best}$, then the current solution is recorded as the best solution and $E_{best} = E_{current}$.

ii) If $E_{move} > E_{current}$

The move would cause a deterioration in E. Generate a random number r, where r is distributed U[0,1]. Let $\Delta E = E_{move} - E_{current}$.

If $r < \exp(-\Delta E/T)$ accept the move and set $E_{current} = E_{move}$, otherwise reject it.

Return to step 2 for N_{it} iterations at the current temperature, T.

Step 4. Revise the current temperature.

After N_{it} iterations cool the current temperature. $T = T \times CR$, $(0 < CR < 1)$.

Step 5. Stopping

Stop when there is no change in the solution for *icount* values of T. (Suresh and Sahu used an icount of 3.)

The performance of the algorithm depends on the values selected for the initial temperature, the cooling ratio and the number of iterations at one temperature. If initial temperature is too low or cooling is too rapid the algorithm may become trapped in a local minimum.

During stochastic U-line balancing as tasks are shifted and crossovers added or deleted the location of all tasks must be recalculated and feasibility of stations checked with respect to P and location constraints. The overhead associated with record keeping and feasibility checks can place extremely heavy computational demands on the algorithm. Algorithm complexity is dramatically reduced if the assumption is made that all task lengths are identical. This would create an ordered system of task locations from the beginning of the U-line to the end, simplifying feasibility checks with respect to P. All distances between the task locations can be calculated prior to solution and location constraints may be expressed as task locations from which crossovers may not originate. Path interference can also easily be checked through using task location numbers,

3.4 The Impact of Exceeding Cycle Time on a U-line.

A U-line station may interact directly with up to four adjacent stations. The effect of exceeding cycle time in a station depends upon its position and the nature of the line. For paced lines with non-removable items, exceeding cycle time in a crossover station may affect two stations by starving them for product or by passing defective products to them. However, the additional interactions may also act to reduce the effect of variability on paced lines with removable items or on unpaced lines. For example in the U-line balance shown in Figure 1, if tasks 3 and 4 in station 2 cannot be completed due to delays in station 1 it may be possible for the operator to process tasks 10 and 11 on the back of the line and then return to the front to complete tasks 3 and 4. In this way the effect of the delay in station 1 may be mitigated to some extent.

4. Unbalancing In Stochastic Assembly Line Balancing

In the deterministic ALB problem, once the minimum number of stations is achieved, a secondary objective of balancing the workload evenly over all stations is frequently included. In stochastic line balancing the interaction between stations may provide an incentive to deliberately unbalance a line to improve overall performance. Hillier and Boling (1966) describe the *bowl phenomenon* where output in a production system with stochastic times is increased through deliberate unbalancing. Mean station times are progressively increased from the centre station to either end of the line. The effect of unbalancing is frequently studied through simulation.

A variety of papers deal with unbalancing as a means of increasing production above the level achieved on a perfectly balanced assembly line. See, for example, Payne et al. [1972], Carnall and Wild [1979], El-Rayah [1979] and Sadowski and Medieros [1979]. In these papers, the extent of improvement in unbalanced systems varied directly with the variability of processing times in final assembly. One criticism of unbalancing studies was that the task time variances used were higher than those typically found in manufacturing environments. Research by Smunt and Perkins [1985] indicated that while unbalancing is advantageous on short lines with high task-time variations, on longer lines with low to moderate variations a balanced line performs best. Other work suggests that the gains from unbalancing are small, generally in the order of 1% (Baker *et al.* [1993]). However, studies by Hillier and Boling [1991] and Baker *et al.* [1993] do suggest that gains above 1% may be achieved if parallel stations are included in the system or if the

coefficient of variation of task times is relatively large.

Villeda et. al. [1988] examined unbalancing in a JIT production system of three lines with three stations each feeding a final assembly area. Their key findings were:

i. Unbalanced production produces higher output than a perfectly balanced system, although the maximum improvement is $< 2\%$.

ii. The best arrangement was a high-medium-low loading arrangement which minimizes the probability of final assembly exceeding cycle time. Since final assembly drives the entire system the effect of variability in this area is amplified throughout the system.

Sarker and Harris [1988] studied the effect of different unbalancing strategies in a six station JIT manufacturing process. Their results differed from those of Villeda et. al., concluding that the best arrangement for a JIT system is a perfectly balanced line. JIT production will operate smoothly if station times fluctuate within a $\pm$ 10% range. They recommended that planned imbalance be incorporated into line balances only to deal with clearly identified issues of variability, such as different operator performance. The results of Villeda et al. and Sarker and Harris appear to be consistent with the findings of Smut and Perkins that unbalancing works best on shorter lines with highly variable task times.

Unbalancing is difficult on a U-line because crossover stations occupy two positions, one at the front and one at the back of the line. For unbalancing purposes both of these positions may require different treatment. The benefits of unbalancing on U-lines are likely to be low since, in JIT production, a major emphasis is placed on the reduction of variability of task processing times. Thus for most U-lines the best solution will be an

evenly balanced line.

5. Discussion

Two different versions of the STULB problem were considered in this paper. In the first, tasks are constrained by the requirement that for every station $Pr(\tau_m < C) \geq \alpha$. This problem may be solved using stochastic dynamic programming. One of the difficulties with this approach is that if consideration of travel and task locations is included the size of the solution space becomes prohibitively large. The more general version of the problem is solved using a heuristic, simulated annealing. Use of SA allows the incorporation of travel considerations. Making the assumption that all task lengths are identical dramatically reduces the calculation and storage requirements for a SA algorithm.

A significant amount of research is required to determine the best objectives and solution algorithms for stochastic U-line balancing. This research would examine the effect of varying SA parameters and the effectiveness of other heuristic techniques. The scenarios where unbalancing can be used to improve output are also of interest. However, the application of unbalancing to U-lines presents a new set of problems since crossover stations may simultaneously occupy segments on both the front and back of the U-line. While the difficulties posed by stochastic U-line balancing are significant, the need to accurately duplicate and analyse actual production scenarios provides a reason for continuing and expanding this research.

References

Arcus, A.L., "COMSOAL: a computer method of sequencing operations for assembly lines," *Int. J. of Prod. Res.*, 1966, v. 4, 4.

Baker K.R., S.G. Powell and D.F. Pyke, "Optimal allocation of work in assembly systems," *Management Science*, 1993, v. 39, 1, 101-106.

Carnall, C.A. and R. Wild, "The location of variable work stations and the performance of production flow lines," *Int. J. of Prod. Res.*, 1976, v. 14, 6.

Carraway, R.L., "A dynamic programming approach to stochastic assembly line balancing," *Int. J. of Prod. Res.*, 1989, v. 35, 4, 459-471.

Driscoll, J. and A.A.A. Abel-Shafi, "A simulation approach to evaluating assembly line balancing solutions," *Int. J. of Prod. Res.*, 1985, v. 23, 5, 975-985.

Eglese, R.W., "Simulated annealing: A tool for OR," *European J. of OR*, 1990, v. 40, 271-281

El-Rayah, T.E., "The efficiency of balanced and unbalanced production lines," *Int. J. of Prod. Res.*, 1979, v. 17, 1, 61-75.

Glover, F. and H.J Greenburg, "New approaches for heuristic search: A bilateral linkage with artificial intelligence," *European J. of OR*, 1989, 39, 119-130.

Hillier, F.S. and Boling, R.N., The effect of some design factors on the efficiency of production lines with variable operations times, *J. of Industrial Engineering*, 1966, 42, 651-658.

Kao, E.P.C., "A preference order dynamic program for stochastic assembly line

balancing," 1976, v. 22, 10, 1097-1104

Kirkpatrick, S., C.D. Gelatt,Jr. and M.P. Veechi, "Optimization by simulated annealing," *Science*, 1983, v. 220, 671-74.

Malakooth, B.B., "Assembly line balancing with buffers by multiple criteria optimization," *Int. J. of Prod. Res.*, 1994, v. 32, 9,2159-2178.

Mitten, L.G., "Preference order dynamic programming," *Mgmt. Sci.*, 1974, v. 21, 1, 43-46.

Moodie, C.L. and H.H. Young, "A heuristic method of assembly line balancing for assumptions of constant or variable work element times," *Journal of Industrial Engineering*, 1965, 16, 1.

Nkasu M.M. and K.H. Leung, "A stochastic approach to assembly line balancing," *Int. J. of Prod. Res.*, 1995, v. 33, 4, 975-991..

Payne, S., N. Slack and R. Wild, "A note on the operating characteristics of 'balance' and 'unbalanced' production flow lines. *Int. J. of Prod. Res.*, 1972, v. 10, 1, 93-98.

Queyranne M., "Bounds for Assembly Line Balancing Heuristics," *Oper. Res.*, 33, 1985, 1353-1359.

Reeve, R.N. and W.H. Thomas, "Balancing stochastic assembly lines," *AIIE Transactions*, 1973, 5, 223-229

Sadowski, R.P. and D.J. Medieros, "The effect of variability on the unbalanced production line," *Int. J. of Prod. Res.*, 1979, v. 17, 6.

Sarker, B.R. and R.D. Harris, "The effect of imbalance in a just-in-time production

system: A simulation study, *Int. J. of Prod. Res.*, 1988, v. 26, 1, 1-18.

Schmidt, L.C. and J. Jackman, "Evaluating assembly sequences for automatic assembly systems," *IIE Transactions*, 1995, v.27, 23-31.

Smunt, T.L. and W.C. Perkins, "Stochastic unpaced line design: reviews and further experimental results," *Journal of Operations Management*, 1985, v. 5, 3

Sniedovich, M., "Analysis of a preference order assembly line problem," *Management Science*, v. 27, 9, 1067-1080.

Sphicas, G.P. and F.N. Silverman, "Deterministic Equivalents for stochastic assembly line balancing," *AIIE Transactions*, 1976, v. 8, 2, 280-282.

Suresh, G., and S. Sahu, Stochastic assembly line balancing using simulated annealing," *Int. J. of Prod. Res.*, 1994, v. 32, 8,1801-10.

Villeda, R., R. Dudek and M. Smith, Increasing the production rate of a just-in-time production system with variable operation times, *Int. J. of Prod. Res.*, 1988, v. 26, 11,1749-68.

Concluding Remarks

The motivation for studying U-line balancing was to determine if there are characteristics of U-lines and their use in JIT production which can provide a competitive advantage to manufacturing organizations. Although the differences between a U-line and a straight line appear minor, this research has demonstrated that they can have a significant impact on manufacturing.

The investigation began with an indepth study of the problem of balancing a single U-line constrained only by precedence and cycle time constraints, the SULB problem. The lower labour requirements for U-lines compared to straight lines was demonstrated in a computational study. Since the SULB problem is more complex than the straight line balancing problem, $O(N^2[\ \#(S^f)^2])$ compared to $O(N^2\ [\#(S^f)])$, identifying an efficient optimal solution algorithm was essential. Different dynamic programming and integer programming algorithms were tested and an effective depth-first branch and bound algorithm was identified. It served as a component of solution algorithms for more difficult ULB problems.

In Part II the SULB problem was extended to include consideration of operator travel in stations. This extension was necessary for two reasons:

1) The time required for travel in a U-line station is a function of task length, U-line layout and task locations. This time must be calculated to ensure

station feasibility relative to cycle time.

2) Paths for operator travel are not allowed to interfere with adjacent stations. To avoid interference, task locations and travel paths must be recorded and non-interference constraints must be added to the problem formulation.

These changes were incorporated into the SULB-T problem formulation and branch and bound algorithm.

Additional gains in worker productivity may be made if two or more U-lines are balanced simultaneously. The concept of multiple U-line balancing was introduced in Part II through the two U-line balancing with travel problem. Access to each U-line was restricted to the opening of the U and an optimal solution algorithm was proposed. When more than two lines were balanced simultaneously, heuristic solution methods were required. The procedure used depended on whether U-line locations were fixed or not. In Part III the NULB-T problem was modelled and heuristic algorithms were proposed for both the general and fixed location NULB-T problems. A computational study demonstrated that the gains achieved by N U-line balancing compared to single U-line balancing were substantial.

In many JIT facilities several product models are produced on a single line in mixed-model production. Units of different models are interspersed through the production sequence in proportion to their demand. Difficulties in mixed-model production result primarily from model imbalance, the fluctuation in total station times. On U-lines different models are mixed in crossover stations during a single cycle. This provides U-lines with the unique ability to reduce model imbalance to levels

unattainable on straight lines. Algorithms for balancing and smoothing U-line balances were offered for both single and multiple mixed-model U-lines.

In Part V the problem of U-line balancing where task times are stochastic was examined. A preference order dynamic programming algorithm for solving one strongly constrained version of the problem was presented. For more general task time distributions or problem formulations a statistical search technique, simulated annealing, was employed to find a heuristic solution.

Although this research focussed on issues and opportunities related to U-lines its applicability reaches beyond this single configuration. The principles and advantages of crossover stations, multiple line balancing and smoothing mixed-model balances may be expanded into other line configurations where stations could contain non-adjacent tasks including:

- Manufacturing cells,
- Single assembly lines whose shape brings separate line segments close together allowing stations to include tasks on different segments and
- Multiple lines of various layouts operating in close proximity.

Manufacturing efficiency will continue to be a critical factor in the ability of companies to compete in the increasingly competitive global economy. The benefits of using U-lines or of incorporating aspects of U-line balancing into assembly line production are significant and can be an important component of future manufacturing competitiveness.

Extensions To U-Line Balancing Research

Although many different aspects of U-line balancing were examined in this thesis numerous research opportunities remain in the search for greater assembly line efficiency and more accurate depiction of manufacturing situations and difficulties. The purpose of the thesis was to model and investigate the characteristics of different U-line balancing problems and to propose a method of solving each. The efficiency and effectiveness of the algorithms may be improved through further research into the following topics:

Bounds

Because of the flexibility associated with U-line balancing tight upper and lower bounds on the number of stations and operator travel are significant factors in algorithm performance. Although several bounds were introduced in the ULB algorithms, the tightness of the bounds may be improved particularly relative to operator travel. The tradeoff between computational requirements for bound calculation and potential improvements in algorithm performance must be evaluated.

Search procedures.

For problems of typical size (5-20 tasks) the breadth-first B&B algorithm proposed proved effective but solution time was high. Improved methods for depicting the problem, better task numbering schemes and search procedures could contribute to enhanced algorithm performance. The use of simplifying assumptions, such as

assuming all task lengths are equal, can dramatically improve performance at minimal expense in terms of accuracy.

Heuristics

Further research into heuristics is encouraged for two reasons.

1. The heuristics used in this study proved effective for solving ULB problems. This conclusion was supported by the computational studies of Parts I-III.
2. Complex ULB problems cannot be solved optimally.

   The complexity of NULB-T, MMULB and STULB problems prevents optimal solution and dictates the use of heuristic solution procedures.

The development and use of effective single and multiple heuristics will improve both the solutions for difficult U-line balancing problems and the time required to solve ULB problems.

Relaxation of access restrictions for multiple U-line balancing

In the multiple U-line balancing problems considered in this thesis access to U-lines was restricted. In certain environments there will be few or no physical restrictions on access to U-line segments. While the nature of this problem is not significantly different from that of the NULB problem where access is constrained, it will require different solution methods, including rules for setting priorities for task assignments and station layout.

Multiple line balancing on U-lines with unrelated cycle times

In facilities where adjacent lines produce parts for unrelated products, creating multi-line stations is complicated by the fact that the lines generally operate with different cycle times. Careful timing of production in each area of the multi-line stations and the addition of work-in-process inventory will be critical to their successful operation. Such stations would allow an opportunity for further reduction in labour requirements but at the cost of increased manufacturing complexity and work in process inventory. The tradeoff between these benefits and costs must be carefully assessed.

Improved smoothing algorithms for mixed-model U-line balancing

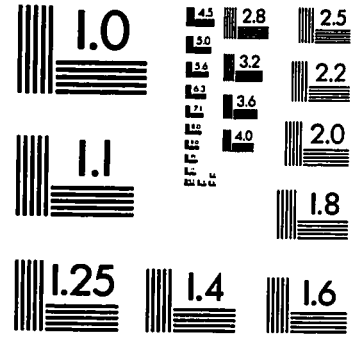
For mixed model line balancing two factors play a role in the success of the final balance: the model imbalance measure employed and the procedure used to smooth the line balance. Further research is required to define effective model imbalance measures, to gauge the effect of model imbalance measure choice and to determine the situations in which each measure should be used. Research into smoothing methods should concentrate on evaluating procedures for selecting stations and tasks to smooth and alternative task exchange rules. Research into the use of statistical search algorithms may prove promising.

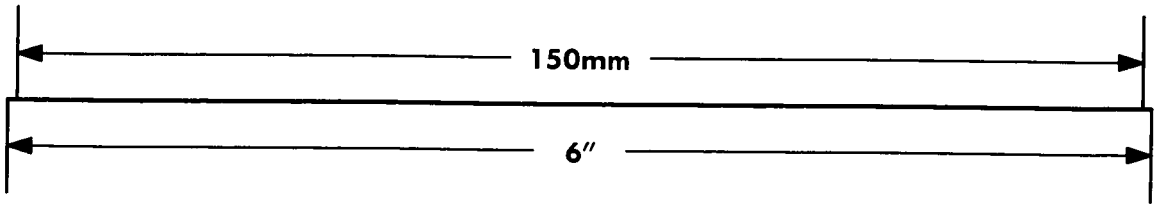
Extending U-line balancing concepts into other line configurations

As discussed in the concluding remarks many advantages of using U-lines may be obtained to some degree in line configurations more common to North American manufacturing. Although U-line methods can improve manufacturing efficiency further research is required to define the situations where such methods may be used

and the conditions which must exist to make their use successful.

Many of these issues will be dealt with in future research.





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989