Scheduling and Lot Streaming
in Two Machine No-Wait Openshops

by

Esaignani Selvarajah

A Thesis Submitted in Conformity with the Requirements for the
Degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

© Copyright by Esaignani Selvarajah (1999)
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-45624-2
Dedicated to my beloved parents
Abstract

Lot streaming is the process of splitting a production lot into a number of sublots, in order to allow overlapping of successive operations, in multi-machine manufacturing systems. The well documented benefits of lot streaming include reductions in lead times and work-in-process, and increases in machine utilization rates. We study the problem of minimizing the makespan in two machine no-wait openshops producing multiple products. We assume that there is a fixed maximum number of sublots for each product, and that lot streaming is possible. This intractable problem requires finding subplot sizes, a product sequence for each machine and a machine sequence for each product. We prove that optimal solutions can always be obtained for single product continuous case in polynomial time. A heuristic algorithm, Global is developed for the multi-product integer-sized subplot problem. For the multi-product case, we simultaneously find the optimal number of sublots from a given upper bound on the number of sublots, sublots sizes and the product sequence. This problem is shown to be equivalent to a Traveling Salesman Problem (TSP) with a pseudo-polynomial number of cities, for which computationally efficient heuristics are tested. Finally we provide some possible extensions of our model and the results of the computational experiments on randomly generated problem instances.

Key Words and Phrases: machine scheduling, no-wait openshops, lot streaming, traveling salesman problem.
Acknowledgments

First and foremost, I express my sincere gratitude to Professor Chelliah Sriskandarajah, who helped me profusely at various stages of this thesis work. His guidance and profound involvement in my research needs appreciation. His support in arranging financial assistance for me warrants special mention.

I gratefully acknowledge the guidance and suggestions provided by Prof. Nicholas G. Hall of Ohio State University. I would like to thank Professor A.K.S Jardine and Professor V.Makis for their valuable comments and suggestions which improved the content and the presentation of my thesis. I am also thankful to Prof. Tapan Bagchi of Indian Institute of Technology Kanpur, Prof. Rogers, Prof. Carter, Prof. M J M Posner and whole MIE family who helped me at various stages.

This research was supported in part by Natural Sciences and Engineering Research Council of Canada (Grant no. OGP0104900). I appreciate NSERC for its fund support to pursue my thesis work. I wish to thank Professor Gilbert Laporte (University of Montreal) for supplying a copy of the GENIUS program.

The administrative assistance provided by Prof. Beno Benhabib as Graduate Coordinator, Norma Dotto, Louisa and Brenda as Departmental staff and Oscar del Rio as computer systems Administrator is deeply remembered.

I greatly acknowledge my friends Mr. Ganesharajah, Mr. Kohulan, Mr. Subodha Kumar, Mr. Thampu Joseph, Mr. Devinder and Mr. Bimal for their invaluable help and support, who made my stay at UofT an enjoyable period. My special thanks are for Mr. Subodha Kumar for his timely assistance. Mr. and Mrs. Raman Patel deserve thanks for the moral support they provided.

Last but not the least I would like to express my deep gratitude to my parents, my brothers, my sisters and my in-laws, who are always a source of motivation and inspiration to me.
# Contents

1 Introduction ................................................. 1

2 Literature Review ........................................... 5

3 Notations and Assumptions .................................. 16

4 Lot Streaming: Single product and continuous sized sublots 18
   4.1 Introduction ........................................... 18
   4.2 Scheduling and Algorithm Development .................. 19
   4.3 Different Structures of Schedule ......................... 38
   4.4 Conclusion ............................................ 39

5 Multiple Products ............................................ 40
   5.1 Introduction ........................................... 40
   5.2 Scheduling and Algorithm Development .................. 41
      5.2.1 Scheduling of Products with Given Profiles ......... 41
      5.2.2 Simultaneous Lot Streaming and Scheduling ......... 42
   5.3 Conclusion ............................................ 50

6 Computational Testing ....................................... 52

7 Some Related Results and Discussion ....................... 54
   7.1 Single Product Integer-sized Sublots .................... 54
   7.2 Problems with Attached Setup Times ...................... 54
   7.3 Sublots with Restricted Capacity ......................... 55
   7.4 Problems with Sublot Transfer Time ...................... 55
7.5 Two Machine No-Wait Flowshop

8 Conclusions
List of Figures

2.1 A basic flowshop model with 4 machines processing one lot. ............... 7
2.2 A basic flowshop model with 4 machines processing two sublots. .......... 7
2.3 Network representation of a single job m machine flow shop problem .... 10
2.4 Time lag model for a given job on two machine problem ................... 12
4.1 A busy schedule for Case 1. a = b, n = 2k, \( C_{\text{max}} = s + Wa \). ........ 20
4.2 A busy schedule for Case 2.1. a > b, \( s \geq t \), n = 2k, \( C_{\text{max}} = s + Wa \). 21
4.3 A typical schedule of sublots \( X_{2r-1} \) and \( X_{2r} \). .................... 23
4.4 A busy schedule for Case 2.2.1. a > b, \( s < t \), \( Wa + s > Wb + t \), n = 2k, \( C_{\text{max}} = s + Wa \). ......................................... 25
4.5 The busy schedule for Case 2.2.2. a > b, \( s < t \), \( Wa + s = Wb + t \), \( C_{\text{max}} = Wa + s \). .................................................. 29
4.6 A busy schedule for Case 2.2.3. a > b, \( s < t \), \( Wa + s < Wb + t \), n = 2k, \( C_{\text{max}} = s + Wa \). .................................................. 37
4.7 Alternating Pair Structure. .......................................................... 38
4.8 Flowshop Structure. ................................................................. 38
4.9 Hybrid Structure. ......................................................................... 39
5.1 Profiles of Openshop No-wait Lot Streaming Problems. ......................... 41
5.2 Example Schedule of N Products in Sequence \( \sigma \). ................................. 43
5.3 Example With Two Countries and Five Cities. ...................................... 44
5.4 Schedule from Solution 1. ............................................................... 47
5.5 Schedule from Solution 2. .................................................................... 48
5.6 Schedule from Solution 3. .................................................................... 48
5.7 Schedule from Solution 4. .................................................................... 48
5.8 Schedule from Solution 5. ................................................. 49
5.9 Schedule from Solution 6. ................................................. 49
5.10 Optimal Schedule from Solution 7. ............................... 49

7.1 Forward flowshop structure with transfer time. ................. 55
7.2 Backward flowshop structure with transfer time. ............... 56
7.3 Alternating pair structure with transfer time. ................. 56
Chapter 1

Introduction

Scheduling, a decision making process, involves allocation of limited resources such as equipment, labor and space to jobs, activities, tasks, or customers through time. In a manufacturing environment it includes worker and machine assignment, job sequencing, and the coordination of material handling and maintenance support. Effectiveness of a scheduling process is measured by performance measures on the production system. There have been many measures used depending on the organization's objective. One of these performance measures is makespan which is the total time required to process all the jobs on all machines.

The scheduling problem differs considerably based on the type of operation such as continuous production, mass production and project works. Further, it depends on machine environment and technological constraints. These different scheduling problems naturally lead to different models.

The basic machine environments of a shop are flowshop, openshop and jobshop. Flowshop is a special case, where all the jobs follow the same processing order, whereas in jobshop there is no limitations on processing order and a machine can process more than one tasks for a job. Openshop is a special case of jobshop where a machine can process only one task for a job.

Some of the technological constraints in manufacturing systems are precedent requirements, blocking production and no-wait production. Precedent requirements are physical restrictions in the order in which operations are performed. In blocking production system
no buffer exists between machines. Therefore, jobs have to wait on the machine till the succeeding machine is free. No-wait production is basically a blocking production system except a job cannot wait on any machine in no-wait production system.

In mass production case, items are produced in lots usually called economic lot. Lot streaming problems consider splitting of these lots (jobs) into smaller sublots so that machines idle time and the completion time of the lot is minimized. As in many cases, there are different versions of lot streaming problems that exist according to the sublots' nature. In discrete version sublots can have only integer number of items, whereas in real version a sublot can have fraction of items. Further more, if the sublot sizes vary between machines, it is called the variable version, and if the sublot sizes are maintained across the machines, it is called the consistent sublots. There is one more possible version regarding the lot process. When sublots of a given lot are processed continuously, it is called continuous processing.

In addition to the advantage of minimum machine idle time and minimum average inventory, lot streaming makes many organizations cope with the modern manufacturing technique of JIT, where each sublot can be treated as a kanban card. Further, lot streaming can be accommodated in traditional production planning and control models without any difficulties. These inherent merits of lot streaming have attracted many recent researchers.

This thesis studies a production environment with two machines, where products are produced in lots and a lot must be processed from start to finish without any interruption on or between machines. Further, each product requires two operations which can be done in any order. Thus, this system can be referred to as a two machine no-wait openshop mass productions system. We try to apply lot streaming technique on this production system so that the makespan can be minimized. For the single product case, we try to use all the number of sublots available, if possible. In the multi-product case, for each product with its lot size, we find the optimal sublot sizes, the number of sublots from a given upper bound on the number of sublots and the job sequence that would minimize the makespan. The no-wait constraint restricts the sublots to be consistent across the machines. This problem is motivated from a real world problem encountered in integrated
chip manufacturing process in semi-conductor industries, where the chips undergo a final testing at high temperature in lots loaded on boards by automated testing equipments. The boards are specific for most of the cases. There are different testings have to be done. The order of testing is not specific. Further, the time required to test a lot depends on the number of chips and the type of chips on it. The temperature requirement, the nonspecific order of testing and the board specification for the chips demonstrate this problem as a no-wait openshop lot streaming problem.

The problem of minimizing makespan in a two machine no-wait openshop is known to be unary NP-hard (Sahni and Cho 1979). This result also applies to our problem if only one subplot is allowed for each product. Therefore, lot streaming in openshop no-wait problems with one subplot is unary hard. Thus, it is unlikely that our problem can be solved by an optimal efficient algorithm.

This thesis is organized as follows. We give a brief survey of literature on lot streaming in Chapter 2. The notations and assumptions of our model appear in Chapter 3. Chapter 4 studies, the single product continuous case with given maximum number of sublots. The analysis on continuous-sized sublots may give some insights for integer-sized sublots. Further, the analysis can be applied to integer problems when the number of items in a lot is so large and the number of sublots are few. This chapter contains scheduling and algorithm developments, and definitions of several possible subplot structures for scheduling a single product. These definitions are also useful in considering multiple products. We consider the single product continuous case into different cases and provide algorithms which give optimal solutions for each case. We show that equal-sized sublots will not always be optimal, but the equal-sized pairs will be optimal in most of the cases. Even though it is the maximum number of sublots, we assign maximum number of sublots, if possible. Integer case is difficult to solve. However the dynamic programming algorithm, which is provided for the multiple product integer case in Chapter 5, can also be utilized with some modifications to solve the single product problem requiring integer subplot sizes.

Chapter 5 shows that the multiple product problem can be formulated as a large traveling salesman problem (TSP) for which the number of sublots for each product is a
decision variable. The optimal schedule for individual products may not be optimal for the multiple product case. Therefore for each product, we consider all possible schedules using dynamic programming algorithm. Then each product is considered as a country and the profiles of a product are considered as cities of the country. Now it can be treated as a special case of traveling salesman problem where salesman has to visit only one city in a country. We develop a cost matrix to restrict the travel only to one city in a country.

In Chapter 6 we provide detailed computational tests of our algorithm's ability to solve the scheduling and lot streaming problem and in Chapter 7, we discuss some possible extensions that can be derived from our algorithm Global. Chapter 8 contains a conclusion and some suggestions for future research. The "C" coded dynamic programming algorithm, a sample output of the dynamic programming and the TSP algorithm, and the schedule of the output are given in the Appendix.
Chapter 2

Literature Review

In manufacturing organizations, job (product, lot) scheduling is one of the most important tasks. Two major factors to be considered in scheduling are demand for the product and the resources (machines) availability. Usually schedulers assume, no machine can process more than one job at a time. Each job requires different machines to process in different or in the same order depending on the shop environment. Hence the scheduling problem is to schedule jobs such that each machine works on at most one job at a time.

Two major objectives of any manufacturing organizations are customer satisfaction and profit maximization. They both are generally interrelated, i.e., increase in customer satisfaction results increase in turnover. The high turnover leads to lower unit cost due to economies of scale, and gives high profit margin. Thus, the lower unit cost and high sales volume booms the profit. On realizing this fact most of the organizations try to satisfy customer requirements as much as possible to survive in the competitive market environment. High quality products at lower prices, and delivery within due dates are the critical requirements to be satisfied. Therefore, in order to meet customer requirements, products must be available when they are required.

However, producing a product as and when required will not be desirable if some machines are commonly used by many jobs each of which has a high volume of demand, and the setup time for each product on machines are high. Then researchers proposed economic lot size for each product so that the total production and inventory cost is minimized. In traditional lot production, a lot is indivisible and once an operation is started on a lot,
the full lot is completed before it is transformed to its successor operation.

Even though the economic lot size yield minimum total production and inventory cost, it has its own demerits. The economic lot production needs relatively longer time to finish the lot. Therefore, product lead times are high which in turn requires high safety stock. As a result average inventory is increased. If the lot size is small, the lead time will be reduced, but the setup cost will be increased.

In a multi-machine production environment, if an economic lot which consists of a number of identical items (units), can be split into smaller sublots and each subplot is transferred to its successor operation immediately upon completion, the lead time will be reduced while maintaining the minimum setup cost. This technique of splitting a lot into smaller sublots to allow overlapping of operations so that the completion time of a job is minimized is called lot streaming. Even if the overall criterion is something other than the makespan, makespan minimization of individual jobs improves performance. We see, how lot streaming minimizes the makespan, through an example on single job flowshop model. If a job consists of 100 items and there are four machines with unit processing time requirements of 8,3,6 and 4 time units, the job completion time will be 2100 time units if no lot streaming is allowed (Figure 2.1). However, if the job is split into two equal sublots, the completion time will be 1450 time units which is shown in Figure 2.2. It is a 30% reduction in the time requirements. If the number of sublots are increased, the percentage reduction will be increased. However, the improvement is diminishing with the number of sublots. Baker and Jia (1993) test the makespan improvement as a function of the number of sublots for different versions of the lot streaming problem for three machine problems.

Lot streaming makes many organizations cope with the modern manufacturing technique JIT where each subplot can be treated as a kanban card. Further, lot streaming can be accommodated in traditional production planning and control models without any difficulty. For example, in the MRP system, order release date of sublots can be estimated as usual. The inherent advantages of lot streaming has attracted many recent researchers. Unlike preemption of job, lot streaming deals with a particular job. There have been many solution methods proposed by different researchers with their own assumptions. Before
Figure 2.1: A basic flowshop model with 4 machines processing one lot.

Figure 2.2: A basic flowshop model with 4 machines processing two sublots.
discussing the solution techniques of different researchers, we give different versions of lot streaming problems.

In most of the practical cases, sublots are assigned for integer number of items which is called the discrete version of the lot streaming problem. In the continuous version, sublots are assigned for real number of items. Generally, problems of discrete version can be formulated as an Integer Linear Program (ILP). However, solving ILP is difficult with large number of variables. Therefore, researchers solve for the continuous version and round off the solutions using heuristics to solve for the discrete version. When the lot size is very large with small number of sublots, rounding the solution obtained in continuous version will be a reasonable solution. In addition, the optimal makespan in continuous version serves as a lower bound on the optimal makespan in the discrete version, and the makespan found by rounding the continuous solutions serves as an upper bound. Thus, the analysis on continuous version gives some insights on discrete version.

In general, sublot sizes vary between machines and which is called variable sublots. However, there are many technological constraints that may affect the formation and movement of sublots. For example, in a no-wait production system, any item in a sublot cannot wait on a machine or for a machine. Hence the sublots are restricted to maintain the same items across the machines which is called the consistent sublots. Further, consistent sublots are easy for analysis.

In most of the cases, sublots of a given job are processed continuously and it is called continuous processing. In this case, only one setup for each batch is required. When the setup is separable from the job, it is called detached setup. Detached setup can be done once the machine becomes available. However, if the setup is not separable from the job, i.e., attached setup, setup can be done only when the job is available. There have been many lot streaming problems discussed in the literature with different combinations of the above versions.

The goal of the basic lot streaming problem is to find the optimal size of each of a prede-termined number of sublots that will minimize the objective function. The lot streaming problems, however, are difficult to handle mathematically because of the existence of a
large number of transfer lots and the complicated interaction among lots, transfer lots and machines. Lot streaming will not improve the makespan or any other regular performance measures when there is only a single machine production process. Therefore, the simplest lot streaming problem arises with two machine production processes and most of the researchers study two machine cases to get the insight of lot streaming and extend the results obtained, if possible, to more than two machine cases. It is not always possible to reduce the makespan by lot streaming even for more than one machine. Sen and Benli (1998) study two machine multiple job openshop problem and prove there will be improvement on makespan due to lot streaming only if the total processing time of any job on machine 1 and machine 2 is greater than the total processing time of all the jobs either on machine 1 or on machine 2.

Lot streaming approach reduces the average inventory in the system. Hence, the traditional Economic Production quantity (EPQ) will not yield minimum cost. Most of the works on lot streaming assume a given lot size (EPQ) and fixed number of sublots. Szendrovits (1975) develops a model to estimate EPQ and the cycle time, which is the time required to manufacture the complete lot, for a single job continuous processing with equal sublot sizes in a flowshop production system. If the assumption of equal sublot sizes is removed then the makespan may be reduced. Goyal (1976) Studies this model and find the optimal sublot sizes.

Often in practice, the sublot size is kept the same across all the machines to preserve the integrity of sublots throughout the process. Consistent sublots do not always yield the minimum makespan except in some special cases. Potts and Baker (1989) prove consistent sublots of single job in two machine flowshop and three machine flowshop always yield the minimum makespan. They prove that for the two machine flowshop problems all the sublots of optimal sizes are geometric, i.e., $X_{j+1} = pX_j$, where $X_i$ is the size of the $i^{th}$ sublot and $p$ is the ratio between time of processing a unit on machine 2 and machine 1.

Generally the number of sublots are determined by the technological constraints such as the number of pallets available. Thus most of the problems are studied on fixed number of sublots. Lot streaming problems are further restricted by transporter capacity. There
are few papers which analyze the lot streaming problem with limited transporter capacity. Trietsch and Baker (1993) obtain an expression for optimal sublot sizes for the two machine flowshop problem of continuous version with limited transporter capacity and provide an algorithm for the discrete version.

Different mathematical tools and/or existing scheduling theories are used by different researchers. Linear programming model and network model are widely used mathematical tools. Lot streaming problems in flowshops can be explained by a network. For example a single job flowshop lot streaming problem can be represented by a network shown in Figure 2.3, which contains a vertex \((i, j)\) with weight \(p_i X_j\), for each sublot \(X_j\) on every machine \(M_i\), where \(p_i\) is the unit processing time on machine \(M_i\). The directed arc from vertex \((i, j)\) to vertex \((i + 1, j)\) restricts processing of sublot \(j\) on machine \(M_{i+1}\) only after completion on \(M_i\). The directed arc from vertex \((i, j)\) to vertex \((i, j + 1)\) restricts to start sublot \((j + 1)\) on machine \(M_i\) only after the completion of sublot \(j\) on it. Makespan is given by the length of the longest path (critical path) in the network, where the length of any path is the sum of the weights of the vertices on it. Any sublot corresponding to a vertex on the critical path is called the critical sublot.

In a flowshop all jobs follow the same machine sequence, and each job has exactly one operation on each machine. An openshop is similar to a flowshop, except that the operations of a job may be performed in any order. Therefore in openshop problems,
we have to find the machine sequence in addition to the job sequence. A brief survey of openshop scheduling problems can be found in Kubiak et al. (1991), Graham et al. (1979) and Lawler et al. (1993). Useful general references on machine scheduling problems include the books by Baker (1974) and Pinedo (1995).

Glass, Gupta and Potts (1994) use network representation to solve single product lot streaming problems in three machine flowshop and openshop problems and two machine jobshop problems. They claim that their schedule possesses the no-wait and/or blocking property too. Initially, three machine flowshop problem is solved and the result is used to get expressions for two machine jobshop problems. In two machine jobshop problem with three operations, first and third operations of a job are done on the same machine. Therefore, an additional constraint is inserted to prevent simultaneous first and third operations of the same sublot. In the openshop case, routing of items through machines must be specified in addition to the sublot sizes. They prove that equal sublots of size $\frac{1}{m}$ for the first $m$ sublots (when the number of sublots $s$ is not less than the number of machines $m$) and assigns zero items to the remaining sublots and the routing of sublot $i$ as $(i, ..., m, 1, ..., i - 1)$ will give optimal solution. Hence when the number of sublots is not less than three, their model can be used to solve (two- and) three machine lot streaming problem. They further, explain that the lot streaming of equal-sized two sublots, $m$ machine problem can be treated as $m$ sublots and two machine problem which can be solved by the algorithm of Gonzalez and Sahni (1976) to get optimal sequencing. Thus, their models solve three machine lot streaming problems.

Potts and Baker (1989) prove that for the two machine flowshop problems all the sublots of optimal solution are critical in the network model. Chen and Steiner (1998, 1995 and 1996) use network models to solve lot streaming problems with discrete version of the single job in two machine flowshops, three machine flowshop with detached setup and three machine flowshops with attached setup respectively.

Baker (1993) utilizes the existing theory for time lags and setup times to analyze lot streaming of equal sublots in two machine flowshop with setup times. In a two machine
problem with time lag, the processing of a job \((j)\) can begin at machine 2 while the later portion is being carried out on machine 1 as shown in Figure 2.4. \(d_j\) is defined as the transfer lag. From Figure 2.4, we found that there is an overlapping of operations. Therefore, it can be treated for lot streaming problems. Initially, for a two machine flowshop problem, the author finds the relationship between the Johnson algorithm (1954) and the time lag model. Then he uses the result of this problem for the lot streaming problems. He further discusses the extensions to more than two machines using the non-bottleneck property used by Johnson for three machine flowshop problem.

Usually, we assume there is no time to transfer a sublot from one machine to the other. However, if the transfer times are large, then ignoring the transfer time will lead to some erroneous decision. Vickson (1995) provides a Linear Programming (LP) formulation for the lot streaming problem of multiple products in a two machine flowshop with setup times and sublot transfer time. He provides mathematical models for optimal lot streaming and sequencing problems for continuous version with attached setup time/detached setup times and/or lot transfer times when regular performance measures are used, where regular measure is a performance measure which is non-decreasing in the completion time. Finally, the author presents an algorithm for the integer version and illustrate it with an example.

Stephane and Lasserre (1997) provide a mathematical model to solve lot streaming problem with consistent sublots in general jobshop environment for a given number of sublots. They claim this formulation can solve only small instances. Therefore, a two-
level iterative procedure is proposed in order to determine the limits of a global approach precisely. The upper level solves for finding the subplot sizes for a given sequencing using a Linear Programming (LP) model, and the lower level solves for sequencing the sublots with sizes found in upper level with another LP model. Then an iterative procedure is used for solving the above model starting with an initial subplot sizes. Since the iterative procedure gives real values for subplot sizes, the authors present a rounding procedure to get integer-sized sublots. In rounding, they round the subplot sizes to the closest integer for all sublots except the last one. Then the last subplot size is found by deducting the total size of all the sublots from the number of items available. They give a mixed integer LP model to solve lot streaming problem when setup costs are involved. A heuristic is used to solve the lot streaming (integer-sized sublots) and scheduling problems. In the heuristic, the setup time is included in the processing time of each subplot, thereby the makespan is overestimated. Using some experimental results on two sets of problems with and without setup times, the authors conclude that small number of sublots is sufficient for jobshop problems without setup.

In batch manufacturing systems, makespan is the cycle time. There are few researchers who study the lot streaming problem to minimize some performance measures other than makespan. Steiner and Truscott(1993) consider the discrete version of lot streaming problem of multi-machine batch processing system with equal-sized sublots and continuous processing (one setup for each batch) with the objective of minimizing cycle time, flow time (the average length of the time periods during which units are in the system) and/or the total processing cost (sum of work-in-process inventory carrying cost and the subplot transportation cost). They obtain lower bounds for the objective value. If there is no technological constraint, they argue, that unit size sublots will give optimal solution.

We study lot streaming and scheduling problem in openshop environment. In our openshop model, processing of sublots is no-wait in the sense that the processing of a subplot on a machine is commenced as soon as the processing of the subplot on the preceding machine is finished. This may lead to machine idle times between successive sublots. In
a no-wait shop, each sublot must be processed continuously from its start in the first machine, to its completion in the last machine, without any interruption on machines and without any waiting in between the machines. Therefore, no-wait lot streaming problems require consistent sublots.

The characteristics of the processing technology, or the absence of storage capacity between operations of a job may cause no-wait production environments. The scheduling problems in no-wait shop arise in the chemical processing and petrochemical production environment. Another example of no-wait situation arises in hot metal rolling industries, where the metals have to be processed continuously at high temperature.

A considerable amount of interest has arisen in no-wait scheduling problems in the recent years. Refer to the survey papers on no-wait scheduling by Hall and Sriskandarajah (1995) and Goyal and Sriskandarajah (1988). This interest appears to be motivated by applications as well as by questions of research interest.

Let us consider the scheduling problems without lot streaming. Note that minimizing makespan in two machine no-wait openshop shown to be NP-Hard (Sahni and Cho 1979). Thus, it is unlikely that the problem can be solved by optimal efficient algorithm (Garey and Johnson 1978). More closely related to this paper are studies of scheduling problems in no-wait flowshops. Gilmore and Gomory (1964) provide an efficient algorithm for the two machine no-wait flowshop. Röck (1984) shows that the similar problem with three machines is intractable.

Hsu and Stein, (1991) study a real world no-wait problem in chemical manufacturing system called “anodizing line” which is made up of a series of chemical processing tanks. The line produces many types of products for the commercial and automotive industries such as pipes, trim and truck grilles. The objective here is to produce the daily production requirements of various products in the shortest possible time. In front of the line is a racking area, where items of a product are loaded onto racks prior to chemical processing. Since the shapes of products are different, each product has its own racks. The number
of racks available for a product is limited. The products need to be processed as no-wait for two reasons: (i) there is no buffer between tanks (ii) once a rack exits a tank it must immediately enter the next one or the products will be ruined due to deterioration of the items while exposed to the atmosphere. Since it is a chemical process, the processing time of a rack (or a subplot) in a tank depends on the total surface area. Hence the processing time is proportional to the number of items in a rack. There is a setup time to change the chemical compositions of tanks when changing from the processing of one product to another product.

Sriskandarajah and Wagneur (1999) study lot streaming in two machine no-wait flowshops with multiple products. An LP model is formulated to find the optimal subplot sizes for the single product case. They use Gilmore and Gomory algorithm to get optimal sequencing of multiple products with given subplot sizes and prove that the optimal subplot size of a single product will be optimal for the simultaneous problem of lot streaming and sequencing of multiple products. They provide heuristics for integer version of single product and multiple product cases, and carryout computational experiments for the performance evaluation of their heuristic algorithms. Finally, a tabu search technique is given for the simultaneous problem of finding the number of sublots, integer subplot size and sequencing of multiple products.

No studies on lot streaming in no-wait openshop is found in the literature. This is the topic for my thesis project which will be described in the next chapters.
Chapter 3
Notations and Assumptions

Data
$M_1, M_2$: the two machines in the openshop.
$P_j, j = 1, \ldots, N$: the products to be produced.
$a_j, b_j > 0$: the processing times of unit product $P_j$ on machines $M_1$ and $M_2$, respectively.
$n_j$: the maximum allowable number of sublots for product $P_j$.
$W_j$: the number of units demanded for product $P_j$.
$s_j$: the setup time of product $P_j$ on $M_1$.
$t_j$: the setup time of product $P_j$ on $M_2$.

Variables
$X_{ij}$: the total number of units of product $P_j$ in subplot $i$, where $X_{ij} \geq 1$ and integer.
$\mu_j$: the optimal number of sublots for product $P_j$ (a decision variable $2 \leq \mu_j \leq n_j$).

Assumptions
1. All products are available at time zero.
2. Each product has a specified maximum number of sublots.
3. The processing of sublots is no-wait.
4. A product’s sublots are consistent.
5. Setup times are independent of the product sequence, and setup times are attached (in attached setup times, setup can be done only when the product is available).
6. Setup times and processing times on machines are deterministic. There is no breakdown of machines.

7. There is no limitation on the pallet capacity, i.e., there is no limitation on the $X_{kj}$ values.

Assumption 2 specifies that the maximum number of sublots for each product is fixed. This assumption is valid for a production environment in which the number of pallets is limited, and each product is allocated a specified number of pallets to transport the sublots.

Assumption 4 is a necessary condition in no-wait scheduling since in no-wait scheduling items are not allowed to wait on machines or in between machines.
Chapter 4
Lot Streaming: Single product and continuous sized sublots

4.1 Introduction

The analysis on continuous-sized sublot may give some insights for integer-sized sublots. Hence we analyze lot streaming of continuous-sized sublots and develop heuristics for integer-sized lot streaming. Further, the analysis of continuous-sized sublot can be applied to real problem, when the number of items in a product lot is so large (and the number of sublots are few) that the product lot can be treated as infinitely divisible or rounding off of the continuous size will not affect the objective value considerably. Note that each product has a specified maximum number of sublots according to assumption 2. This means that the maximum number of sublots is fixed for each product. This assumption is valid for the production environment in which the number of pallets is limited and each product is provided a specified number of pallets to carry the sublots. It is convenient to handle small lots in production environments. Thus, items need to be distributed in the given number of sublots. In order to distribute the items evenly, it is desirable to have positive number of items in each sublot and maintain equal sublots whenever possible. We consider the production case where setup times are attached with the processing and setup is done once at the start of processing.

For convenience, we call the first two sublots the first pair, the next two the second pair, and so on. We can use any even number of sublots for the optimal solution. However in our
derivation, we use all the sublots if the maximum number of sublots is even. Otherwise, we use one less number of sublots. As can be seen later that it is convenient to deal with the problem with even $n$. In the openshop problems, we can have multiple optimal solutions. As can be seen later that the equal-sized sublots will not always be optimal, but in most cases, equal-sized pairs of sublots will give optimal solutions. In equal-sized pairs of sublots, each pair of sublots are assigned the same number of items ($\frac{2W}{n}$). We analyze the problem into three main cases. Case 1 analyses the problems where $a = b$. Case 2 analyses the problems where $a > b$, and Case 3 analyses the problems where $a < b$.

### 4.2 Scheduling and Algorithm Development

In our discussion, we shall denote by $aX_i$ ($bX_i$) both the time for sublot $i$ on machine $M_1$ (machine $M_2$) and the name of the operations itself. The use will be clear from the content.

**Definition** : a busy schedule is an optimal schedule for an even number $n = 2k$ of sublots, and has the following properties:

1. $\rho_1$: $C_{max} = max\{L_1, L_2\}$, where $L_1 = s + a \sum_{i=1}^{n} X_i = s + aW$, and $L_2 = t + b \sum_{i=1}^{n} X_i = t + bW$.

2. $\rho_2$: There are $k$ time points $\tau_1, \ldots, \tau_k$ such that for $1 \leq i \leq k$, operations $aX_2i-1$ and $bX_{2i}$ end at time $\tau_i$.

$\rho_1$ is clearly a sufficient condition for optimality. It is easy to see that it is also necessary for optimality, because for the optimal makespan, either machine 1 or machine 2 must not have any idle time. The schedule structure satisfying the property $\rho_2$ is called $\rho_2$ structure (or alternating pair schedule). There are two ways to schedule sublots, single routing and multiple routing. In single routing all sublots of a given job follow the same machine sequence. Thus, it represents a flowshop structure. Then the optimal sublot sizes are geometric as given by Potts and Baker (1989) and we cannot get equal-sized or equal-sized pair sublots in any case unless $a = b$. In multiple routing sublots of a job can follow any machine sequence. $\rho_2$ structure lies in multiple routing. Because of this multiple routing
it has the flexibility to get equal sublots sizes or equal pair sizes in many cases. Reader can see this flexibility of the $\rho_2$ structure through out this thesis. Therefore we introduce the $\rho_2$ structure which tries to satisfy our quality requirement of equal-sized sublots as much as possible. We present algorithms to find the optimal sublot sizes for even number of sublots.

**Theorem 1** For a single product and an even number of sublots, there exists an optimal schedule that satisfies both properties $\rho_1$ and $\rho_2$.

Proof: The relative values of $a, b, s$ and $t$ define several cases. Lemma 1 to Lemma 5 prove that the theorem is true for all cases. This is done by deciding the lot sizes and constructing a schedule with property $\rho_2$ so that the property $\rho_1$ is satisfied.

**Case 1** $a = b$.

**Lemma 1** An optimal solution $X^* = (X_1^*, X_2^*, \ldots, X_n^*)$ in Case 1 is given by a busy schedule, where

$$X_i^* = \frac{W}{n}, \quad i = 1, \ldots, n. \quad (4.1)$$

Proof: Since $a = b$, using the sublots $X_i^* = \frac{W}{n}, i = 1, 2, \ldots, n$, a busy schedule can be constructed as in Figure 4.1. Note that there is no idle time on machine 1 or machine 2 during processing. However, there may be some idle time either on machine 1 or machine 2 in the beginning of the schedule which is unavoidable.

Since the properties $\rho_1$ and $\rho_2$ are satisfied, it is a busy schedule and is optimal. □
Figure 4.2: A busy schedule for Case 2.1. \( a > b, s \geq t, n = 2k, C_{\text{max}} = s + Wa. \)

**Case 2: \( a > b. \)**

When \( s \geq t \), no idle time on machine 1 will give an optimal schedule, whereas when \( s < t \), we may need to consider no idle time on machine 1 and/or machine 2 depending on the situation. Hence, we divide Case 2 into further two subcases. Case 2.1 analyses when \( s \geq t \) and Case 2.2 analyses when \( s < t \).

**Case 2.1 \( a > b, s \geq t. \)**

**Lemma 2** An optimal solution \( X^* = (X_1^*, X_2^*, \ldots, X_n^*) \) in Case 2.1 is given by a busy schedule, where

\[
X_i^* = \frac{W}{n}, \quad i = 1, \ldots, n. \tag{4.2}
\]

**Proof:** Using the sublots \( X_i^* = \frac{W}{n}, i = 1, 2, \ldots, n \), a busy schedule as shown in Figure 4.2 can be constructed for Case 2.1.

Let sublots \( r \) and \( (r + 1) \) make a pair. Then,

\[
\frac{a}{b} X_r \geq X_{r+1}, r = 1, 3, \ldots, n - 1
\]

Also \( X_{r+1} \geq \frac{b}{a} X_r, r = 1, 3, \ldots, n - 1 \)

Therefore, \( \frac{b}{a} X_r \leq X_{r+1} \leq \frac{a}{b} X_r \), where \( \frac{b}{a} < 1 \), and \( \frac{a}{b} > 1 \).

If we have \( X_{r+1} = \gamma X_r \), where \( \frac{b}{a} \leq \gamma \leq \frac{a}{b} \), we can have optimal lot sizes. Since \( \gamma = 1 \) falls in the range \( (\frac{b}{a}, \frac{a}{b}) \), equal subplot size will be optimal. Since the schedule satisfies the properties \( \rho_1 \) and \( \rho_2 \) (Figure 4.2), it is a busy schedule and optimal. \( \square \)
Case 2.2: $a > b$, $s < t$.

If $Wa + s > Wb + t$, there should not be any idle time on machine 1 whereas if $Wa + s < Wb + t$ there should not be any idle time on machine 2 in the optimal schedule. When $Wa + s = Wb + t$, both machines must be fully utilized. Therefore, we analyze this case into another three subcases. Case 2.2.1 analyses when $Wa + s > Wb + t$, Case 2.2.2 analyses when $Wa + s = Wb + t$ and Case 2.2.3 analyses when $Wa + s < Wb + t$.

In order to deal with remaining Cases, we define $\Delta_r$ as the difference between time when machine 2 is ready for processing the $r^{th}$ sublot pair and the time when machine 1 is ready for the $r^{th}$ sublot pair in the schedule satisfying $\rho_2$. For example, initially, for the first pair, $\Delta_1 = t - s$, when there is no sublots scheduled but the settings have been performed starting at time zero. Then for the second pair, $\Delta_2 = bX_1 - aX_2$, when the first pairs of sublots have been processed conforming the schedule structure $\rho_2$.

Case 2.2.1 $a > b$, $s < t$, $Wa + s > Wb + t$.

In Case 2.2.1, the lot sizes can be determined by the algorithm Lot1 given below which provides a busy schedule. The algorithm Lot1 assigns items to sublots such that machine 1 is kept busy. This is done in three steps corresponding to different conditions. Thus For the first pair, the setup time of machine 1 and the processing time of the first sublot on machine 1 must be greater than the setup time of machine 2 and the processing time of the second sublot on machine 2. Step 1 therefore, assigns equal-sized sublots if this condition is satisfied. If the first pair satisfies this condition, the remaining pairs will satisfy the condition. The reader can see it in the proof given next to the algorithm. Step 2 checks whether the setup time of machine 1 and the processing time of the first pair on machine 1 is greater than the setup time of machine 2 and the processing time of the first pair on machine 2, if equal-sized pair is used. If this condition is satisfied for the first pair, it will be satisfied for the remaining pairs too which is shown in the proof. Step 3 assigns the first pair so that there is no idle time due to this pair and the remaining sublots satisfy condition for Lemma 2. Since the condition for Lemma 2 is satisfied, it equally assigns the remaining items to the remaining sublots.
Algorithm Lot1

Step 1: If $\frac{W(a-b)}{n} \geq t - s$, then set 
\[ X^*_r = \frac{W}{n}, \quad r = 1, \ldots, n; \] Stop.

Step 2: If \( \frac{2W(a-b)}{n} \geq t - s \), then set 
\[ X^*_1 = \frac{2Wb}{n(a+b)} + \frac{t-s}{a+b}, \quad X^*_2 = \frac{2Wa}{n(a+b)} - \frac{t-s}{a+b}; \]
\[ X^*_r = \frac{W - X^*_1 - X^*_2}{n-2} = \frac{W}{n}, \quad r = 3, 4, \ldots, n; \] Stop.

Step 3: Set 
\[ X^*_1 = \frac{(t-s)a}{(a^2-b^2)} \quad X^*_2 = \frac{(t-s)b}{(a^2-b^2)} \]
\[ X^*_r = \frac{W - X^*_1 - X^*_2}{n-2} = \frac{W}{n-2} - \frac{t-s}{(a-b)(n-2)}, \quad r = 3, 4, \ldots, n; \] Stop.

**Lemma 3** An optimal solution $X^* = (X^*_1, X^*_2, \ldots, X^*_n)$, obtained by the algorithm Lot1, in Case 2.2.1 is given by a busy schedule.

Proof: Let the idle time on machine 1 before $aX_{2r-1}$ is started on it be $I_1$ and the idle time on machine 2 before $bX_{2r}$ is started on it be $I_2$. Figure 4.3 describes a typical schedule of sublots $X_{2r-1}$ and $X_{2r}$. We have to show that the algorithm maintains $I_1 = 0$ for all sublot pairs.

The optimal schedule in general for Case 2.2.1 is shown in Figure 4.4.

**Step 1:** When $r = 1$,
\[ I_1 + aX^*_1 = I_2 + bX^*_2 + \Delta_1. \]
\[ I_2 = I_1 + aX^*_1 - bX^*_2 - (t - s). \]

Since equal sized sublots, 
\[ I_2 = I_1 + \frac{W(a-b)}{n} - (t - s). \]

Note that \( \frac{W(a-b)}{n} \geq t - s \)
Therefore, $I_2 \geq I_1$
\[
\Delta_2 = bX_1^* - aX_2^* \\
= -(a-b)\frac{W}{n} \\
< 0
\]

Thus, machine 1 can be fully utilized by setting $I_1 = 0$ for the first pair. Since $\Delta_2 < 0$, from Lemma 2, equal subplot size is possible for the remaining $(n-2)$ sublots.

Step 2: When $r = 1$,
\[
I_2 = I_1 + aX_1^* - bX_2^* - (t-s) \\
= I_1 + a\left[\frac{2Wb}{n(a+b)} + \frac{t-s}{a+b}\right] - b\left[\frac{2Wa}{n(a+b)} - \frac{t-s}{a+b}\right] - (t-s) \\
= I_1
\]
\[
\Delta_2 = bX_1^* - aX_2^* \\
= \frac{2Wb^2}{n(a+b)} + \frac{b(t-s)}{a+b} - \frac{2Wa^2}{n(a+b)} + a(t-s) \\
= (t-s) - \frac{2W(a-b)}{n}
\]

Since $\frac{2W(a-b)}{n} \geq t-s$
\[
\Delta_2 \leq 0
\]

Since $I_1 = 0$ is possible, machine 1 can be fully utilized. Further, $\Delta_2 \leq 0$. Therefore, from Lemma 2, we can have equal size for the remaining sublots.

Note that $W(a-b) > t-s$ (condition for Case 2.2.1). Therefore subplot sizes for the case $n = 2$ is decided at least in Step 2. Hence the condition in Step 3 may occur for the case with $n > 2$.

Step 3: When $r = 1$,
\[
I_2 = I_1 + aX_1^* - bX_2^* - (t-s) \\
= I_1 + a\left(\frac{(t-s)a}{a^2 - b^2}\right) - b\left(\frac{(t-s)b}{a^2 - b^2}\right) - (t-s) \\
= I_1
\]
Therefore machine 1 can be fully utilized by setting $I_1 = 0$ for the first pair. Since $\Delta_2 = bX_1^* - aX_2^* = 0$, from Lemma 2, equal subplot size can be optimal for the remaining sublots. Further, we need to show that the total items allocated for the first two sublots are strictly less than $W$ and all the items are assigned.

\[
W(a - b) > t - s
\]
\[
W > \frac{t - s}{a - b}
\]
\[
W > \frac{(t-s)(a+b)}{(a-b)(a+b)}
\]
\[
W > \frac{(t-s)a + (t-s)b}{a^2 - b^2}
\]

Therefore, $W > X_1^* + X_2^*$

Number of items assigned
\[
= X_1^* + X_2^* + \sum_{r=3}^{n} \left( \frac{W}{n-2} - \frac{t-s}{(a-b)(n-2)} \right)
\]
\[
= \frac{(t-s)(a+b)}{a^2 - b^2} + W - \frac{t-s}{a-b}
\]
\[
= W
\]

Case 2.2.2 $a > b, s < t, Wa + s = Wb + t$

The optimal solution of Case 2.2.2 should maintain both machines busy. Figure 4.5 shows an optimal schedule for the Case 2.2.2. Algorithm Lot2 provides a busy schedule for this case. Since both machines have to be fully utilized, equal-sized sublots is not possible. Therefore, the algorithm tries to assign equal pair size, if possible. We define another term $\Gamma_r$ as the number of items left to be assigned once the $r^{th}$ pair is scheduled. When the $r^{th}$ pair is to be scheduled, there will be $\Gamma_{r-1}$ items remaining to be assigned to
the \((2k - 2r + 2)\) sublots. The algorithm \text{Lot2} decides lot sizes for the construction of \(\rho_2\) type schedule such that both machine 1 and machine 2 are busy throughout. This is done in two steps corresponding to different conditions. When the \((r - 1)^{th}\) pair is scheduled, Step 2 tries to assign equal-sized subplot pairs. This step is similar to Step 3 in algorithm \text{Lot1}. It can be recalled that \(\frac{a}{2} = k\). If the equal pair is not possible for the \(r^{th}\) pair, Step 3 assigns the \(r^{th}\) pair such that both machines are busy and the total number of items is less than the total number of items available when the \(r^{th}\) pair is not last. If it is the last pair it selects the sublots such that no idle time is inserted on any machine and all the items are assigned.

\textbf{Algorithm \text{Lot2}}

\begin{itemize}
  \item \textbf{Step 1} \quad \text{Set } r = 1, \Delta_r = t - s, \Gamma_0 = W.
  \item \textbf{Step 2} \quad \text{If } \frac{R_{r+1}}{k-r+1} > \frac{\Delta_r}{a}, \text{ then set}
    \begin{align*}
      X_{2r-1}^* &= \frac{\Gamma_{r+1}}{(k-r+1)(a+b)} + \frac{\Delta_r}{a+b}, \quad X_{2r}^* = \frac{\Gamma_{r+1}}{(k-r+1)(a+b)} - \frac{\Delta_r}{a+b};
    \end{align*}
    \text{If } r = k, \text{ Stop.}
  \item \text{Else, Go to Step 4.}
  \item \textbf{Step 3} \quad \text{If } r < k \text{ then set,}
    \begin{align*}
      X_{2r-1}^* &= \frac{\Delta_r}{2(k^2-b^2)}; \quad X_{2r}^* = \frac{\Delta_r}{2(k^2-b^2)}.
    \end{align*}
    \text{If } r = k \text{ then set,}
    \begin{align*}
      X_{2k-1}^* &= \frac{\Delta_k}{(a^2-b^2)}, \quad X_{2k}^* = \frac{\Delta_k}{(a^2-b^2)}; \quad \text{Stop.}
    \end{align*}
  \item \textbf{Step 4} \quad \text{If } r < k, \text{ then set}
    \begin{align*}
      \Gamma_r &= \Gamma_{r-1} - X_{2r-1}^* - X_{2r}^*, \quad \Delta_{r+1} = bX_{2r-1}^* - aX_{2r}^*.
      r &= r + 1; \quad \text{Go to Step 2.}
    \end{align*}
\end{itemize}

\textbf{Lemma 4} \quad \text{An optimal solution } X^* = (X_1^*, X_2^*, \ldots, X_n^*) \text{ in Case 2.2.2 is given by the algorithm \text{Lot2}.}

\textbf{Proof:} \quad \text{We have to show that in each step } I_1 = I_2 = 0 \text{ is possible for each pair, } \Delta_{k+1} = 0 \text{ and the condition for Case 2.2.2 is hold for all pairs.}
Step 2: Since $W(a - b) = t - s$, the case $n = 2$ will fall in this Step.

$$aX^*_{2r-1} + I_1 = bX^*_{2r} + I_2 + \Delta_r \quad \text{(Figure 4.3)}$$

$$I_1 = bX^*_{2r} - aX^*_{2r-1} + I_2 + \Delta_r$$

$$= \frac{\Gamma_{r-1}(ab - ab)}{(k - r + 1)(a + b)} - \frac{\Delta_r(a + b)}{a + b} + I_2 + \Delta_r$$

$$= I_2$$

Therefore, both machines can be maintained busy by setting $I_1 = I_2 = 0$.

$$\Gamma_r = \Gamma_{r-1} - X^*_{2r-1} - X^*_{2r}$$

$$= \Gamma_{r-1} - \frac{\Gamma_{r-1}}{k - r + 1}$$

$$\Delta_{r+1} = bX^*_{2r-1} - aX^*_{2r}$$

$$= \frac{\Gamma_{r-1}(b^2 - a^2)}{(k - r + 1)(a + b)} + \Delta_r$$

If $\Delta_r = \Gamma_{r-1}(a - b)$,

$$\Delta_{r+1} = \frac{\Gamma_{r-1}(b^2 - a^2)}{(k - r + 1)(a + b)} + \Gamma_{r-1}(a - b)$$

$$= \Gamma_{r-1}(a - b) - \frac{\Gamma_{r-1}}{k - r + 1}(a - b)$$

$$= (\Gamma_{r-1} - \frac{\Gamma_{r-1}}{k - r + 1})(a - b)$$

$$= \Gamma_r(a - b)$$

Therefore, $\Delta_{r+1} = \Gamma_r(a - b)$ will be true, if $\Delta_r = \Gamma_{r-1}(a - b)$. We are given $\Delta_1 = \Gamma_0(a - b)$. Hence by mathematical induction, all remaining sublots and the remaining items will satisfy the condition for Case 2.2.2.

When $r = k$,

$$\Delta_{k+1} = \Delta_k - \Gamma_{k-1}(a - b)$$

Since $\Delta_k = \Gamma_{k-1}(a - b)$,

$$\Delta_{k+1} = 0$$
Step 3: Case $r < k$

\[
I_1 = bX_{2r}^* - aX_{2r-1}^* + I_2 + \Delta_r \\
= \frac{\Delta_r b^2}{2(a^2 - b^2)} - \frac{\Delta_r (2a^2 - b^2)}{2(a^2 - b^2)} + I_2 + \Delta_r \\
= \frac{\Delta_r 2b^2 - a^2}{2(a^2 - b^2)} + I_2 + \Delta_r \\
= I_2 
\]

Therefore, both machines can be kept busy by setting $I_1 = I_2 = 0$.

\[
X_{2r-1}^* + X_{2r}^* = \frac{\Delta_r (2a^2 - b^2 + ab)}{2a(a^2 - b^2)} \
= \frac{\Delta_r (2a - b)}{2a(a - b)}
\]

If $\Delta_r = \Gamma_{r-1}(a - b)$

\[
X_{2r-1}^* + X_{2r}^* = \frac{\Gamma_{r-1} (2a - b)}{2a}
\]

\[
\Gamma_r = \Gamma_{r-1} - X_i^* - X_{i+1}^*
\]

\[
= \Gamma_{r-1} - \frac{\Gamma_{r-1} (2a - b)}{2a}
\]

\[
= \frac{\Gamma_{r-1} b}{2a}
\]

\[
\Delta_{r+1} = bX_{2r-1}^* - aX_{2r}^*
\]

\[
= \frac{\Delta_r (2a^2 - b^2)b}{2a(a^2 - b^2)} - \frac{\Delta_r ab}{2(a^2 - b^2)}
\]

\[
= \frac{\Delta_r (2a^2b - b^3 - a^2b)}{2a(a^2 - b^2)}
\]

\[
= \frac{\Delta_r b}{2a}
\]

\[
= \frac{\Gamma_{r-1} (a - b) b}{2a}
\]

But \[
\Gamma_r = \frac{\Gamma_{r-1} b}{2a}
\]

Therefore, \[
\Delta_{r+1} = \Gamma_r (a - b)
\]

$\Delta_{r+1} = \Gamma_r (a - b)$ is true, if $\Delta_r = \Gamma_{r-1}(a - b)$. Further $\Gamma_0(a - b) = \Delta_1$. Therefore, by mathematical induction, the condition for Case 2.2.2 is satisfied for all remaining items and subplot pairs.
Step 3: Case \( r = k \)

\[
I_1 = bX_{2k}^* - aX_{2k-1}^* + I_2 + \Delta_k
\]

\[
= \frac{\Delta_k b^2}{(a^2 - b^2)} - \frac{\Delta_k a^2}{(a^2 - b^2)} + I_2 + \Delta_k
\]

\[
= I_2
\]

Therefore, both machines can be kept busy by setting \( I_1 = I_2 = 0 \).

\[
\Delta_{k+1} = bX_{2k-1}^* - aX_{2k}^*
\]

\[
= \frac{\Delta_k ab}{a^2 - b^2} - \frac{\Delta_k ab}{a^2 - b^2}
\]

\[
= 0
\]

\[
X_{2k-1}^* + X_{2k}^* = \frac{\Delta_k (a + b)}{(a^2 - b^2)}
\]

\[
= \frac{\Delta_k}{(a - b)}
\]

but \( \Delta_k = \Gamma_{k-1}(a - b) \)

Therefore, \( X_{2k-1}^* + X_{2k}^* = \Gamma_{k-1} \).

Thus all the items are allocated. \( \Box \)
Case 2.2.3 $a > b, s < t, Wa + s < Wb + t$

The subplot sizes for Case 2.2.3 can be determined by the algorithm Lot3.

**Algorithm Lot3**

**Step 1** Set $r = 1$, $\Delta_r = t - s$, $\Gamma_0 = W$

**Step 2** If $\frac{\Gamma_{r-1}}{k - r + 1} a > \Delta_r$, then set

$$X_{2i-1}^* = \frac{\Gamma_{r-1} b}{(k - r + 1)(a+b)} + \frac{\Delta_r}{a+b}, \quad X_{2i}^* = \frac{\Gamma_{r-1} a}{(k - r + 1)(a+b)} - \frac{\Delta_r}{a+b}, \quad i = r, r + 1, ..., k$$

Stop.

**Step 3** If $\frac{\Gamma_{r-1}}{k - r + 1} a > \Gamma_{r-1}(a - b)$, then set

$$X_{2i-1}^* = \frac{\Gamma_{r-1}(a+2b)}{(2k-3r+2)(a+b)} + \frac{\Gamma_{r-1}(a-b)}{2(a+b)}, \quad X_{2i}^* = \frac{\Gamma_{r-1} a}{(2k-2r+2)(a+b)} - \frac{\Gamma_{r-1}(a-b)}{2(a+b)}, \quad i = r, r + 1, ...$$

Stop.

**Step 4** If $\Gamma_{r-1} a > \Delta_r$,

If $r = k$, then set

$$X_{2k-1}^* = \frac{\Gamma_{r-1} b}{a+b}, \quad X_{2k}^* = \frac{\Gamma_{r-1} a}{a+b} - \frac{\Delta_k}{a+b}, \quad \text{Stop.}$$

Else set

$$X_{2r-1}^* = \frac{\Delta_r(a+2b)}{2a(a+b)} + \frac{\Gamma_{r-1}(a-b)}{2(a+b)}, \quad X_{2r}^* = \frac{\Delta_r}{2(a+b)} - \frac{\Gamma_{r-1}(a-b)}{2(a+b)},$$

Go to Step 6.

**Step 5** If $r = k$, then set

$$X_{2k-1}^* = \frac{\Gamma_{r-1} b}{a+b}, \quad X_{2k}^* = \frac{\Gamma_{r-1} a}{a+b}; \quad \text{Stop.}$$

Else set

$$X_{2r-1}^* = \frac{\Gamma_{r-1}(2a^2 - \Delta^2)}{4a(a+b)} + \frac{\Gamma_{r-1}(2a - \Delta)}{4a}, \quad X_{2r}^* = \frac{\Gamma_{r-1} b}{4(a+b)}$$

**Step 6** Set,

$$\Gamma_r = \Gamma_{r-1} - X_{2r-1}^* - X_{2r}^*, \quad \Delta_{r+1} = bX_{2r-1}^* - aX_{2r}^*$$

$r = r + 1$; Go to Step 2.
Lemma 5 An optimal solution $X^* = (X_1^*, X_2^*, \ldots, X_n^*)$, obtained by the algorithm Lot3, in Case 2.2.3 is given by a busy schedule.

Figure 4.3 describes a typical schedule of sublots $X_{2r-1}$ and $X_{2r}$ by algorithm Lot3.

Proof: We have to show that the algorithm maintains $I_2 = 0$ for all pairs, and if a pair falls in Step 2 or Step 3 then the remaining pairs fall in the same step.

Step 2:

\[ I_1 = I_2 - aX_{2r-1}^* + bX_{2r}^* + \Delta_r \]
\[ = I_2 - \frac{\Gamma_{r-1}ab}{(k-r+1)(a+b)} - \frac{\Delta_r a}{a+b} + \frac{\Gamma_{r-1}ab}{(k-r+1)(a+b)} - \frac{\Delta_r b}{a+b} + \Delta_r \]
\[ = I_2 \]

Thus $I_1 = 0$ when $I_2 = 0$. Hence, machine 2 can be fully utilized.

\[ \Delta_{r+1} = bX_{2r-1}^* - aX_{2r}^* \]
\[ = \frac{\Gamma_{r-1}b^2}{(k-r+1)(a+b)} + \frac{\Delta_r b}{a+b} - \frac{\Gamma_{r-1}a^2}{(k-r+1)(a+b)} + \frac{\Delta_r a}{a+b} \]
\[ = -\frac{\Gamma_{r-1}(a^2 - b^2)}{(k-r+1)(a+b)} + \Delta_r \]
\[ = \frac{\Gamma_{r-1}(a-b)}{k-r+1} - \frac{\Delta_r}{k-r+1} \]

If $\Delta_r > \Gamma_{r-1}(a-b)$

\[ \Delta_{r+1} > \Gamma_{r-1}(a-b) - \frac{\Gamma_{r-1}(a-b)}{k-r+1} \]
\[ = (\Gamma_{r-1} - \frac{\Gamma_{r-1}}{k-r+1})(a-b) \]

But, $\Gamma_r = \Gamma_{r-1} - \frac{\Gamma_{r-1}}{k-r+1}$

Therefore $\Delta_{r+1} > \Gamma_r(a-b)$

Further $\Delta_1 > \Gamma_0(a-b)$

Therefore by mathematical induction, remaining items will satisfy the condition for Case 2.2.3.
Therefore, if $\Delta_i < \frac{\Gamma_{r-1}a}{k-r+1}$, then $\Delta_{r+1} < \frac{\Gamma_{r}a}{k-r}$. Further, $\Delta_1 < \frac{\Gamma_{r}a}{k}$. Therefore, by mathematical induction, the remaining items and sublots will satisfy the condition for Step 2.

When $r = k$,

$$X_{2k-1}^* + X_{2k}^* = \frac{\Gamma_{k-1}b}{(a + b)} + \frac{\Delta_k}{a + b} + \frac{\Gamma_{k-1}a}{(a + b)} - \frac{\Delta_k}{a + b}$$

$$= \Gamma_{k-1}$$

Hence all the items are assigned.

$$\Delta_{k+1} = \Delta_k - \Gamma_{k-1}(a - b)$$

$$> 0$$

Step 3:

$$I_1 = I_2 - aX_{2r-1}^* + bX_{2r}^* + \Delta_r$$

$$= I_2 - \frac{\Gamma_{r-1}(a + 2b)a}{(2k - 2r + 2)(a + b)} - \frac{\Gamma_{r-1}(a - b)a}{2(a + b)} + \frac{\Gamma_{r-1}ab}{(2k - 2r + 2)(a + b)} - \frac{\Gamma_{r-1}(a - b)b}{2(a + b)} + \Delta_r$$

$$= I_2 + \frac{\Gamma_{r-1}a(b - a - 2b)}{(2k - 2r + 2)(a + b)} + \frac{\Gamma_{r-1}(a - b)(a + b)}{2(a + b)} + \Delta_r$$

$$= I_2 - \frac{\Gamma_{r-1}a}{(2k - 2r + 2)} + \frac{\Gamma_{r-1}(a - b)}{2} + \Delta_r$$

Since $\frac{\Gamma_{r-1}a}{(k-r+1)} > \Gamma_{r-1}(a-b)$,

$$I_1 > I_2 - \frac{\Gamma_{r-1}a}{(2k - 2r + 2)} + \frac{\Gamma_{r-1}a}{(2k - 2r + 2)} + \Delta_r$$

$$I_1 > I_2 - \frac{\Gamma_{r-1}a}{(k-r+1)} + \Delta_r$$

32
Since \( \frac{\Gamma_{r-1} a}{(k - r + 1)} \leq \Delta_r \)
\[ I_1 > I_2 \]

Hence, machine 2 can be fully utilized.

\[
\Delta_{r+1} = bX_{2r-1}^2 - aX_{2r}^2
= \frac{\Gamma_{r-1}(a + 2b)b}{(2k - 2r + 2)(a + b)} + \frac{\Gamma_{r-1}(a - b)b}{2(a + b)} - \frac{\Gamma_{r-1}a^2}{(2k - 2r + 2)(a + b)} + \frac{\Gamma_{r-1}(a - b)a}{2(a + b)}
= \frac{\Gamma_{r-1}(a - b)}{2} + \frac{\Gamma_{r-1}(a + 2b^2 - a^2)}{(2k - 2r + 2)(a + b)}
= \frac{\Gamma_{r-1}(a - b)}{2} + \frac{\Gamma_{r-1}(2b - a)}{(2k - 2r + 2)}
= \frac{\Gamma_{r-1}(a - b)}{2} - \frac{2\Gamma_{r-1}(a - b)}{(2k - 2r + 2)} + \frac{\Gamma_{r-1}a}{2k - 2r + 2}
\]

But \( \Gamma_r = \Gamma_{r-1} - \frac{\Gamma_{r-1}}{k - r + 1} \)
\[ = \Gamma_r(a - b) - \frac{\Gamma_{r-1}(a - b)}{2} + \frac{\Gamma_{r-1}a}{2k - 2r + 2} \]

If \( \frac{\Gamma_{r-1} a}{k - r + 1} > \Gamma_{r-1}(a - b) \)
\( \Delta_{r+1} > \Gamma_r(a - b) \)

Further, \( \Delta_1 > \Gamma_0(a - b) \)

Therefore, by mathematical induction, condition for Case 2.2.3 is satisfied.

If \( \frac{\Gamma_{r-1} a}{k - r + 1} > \Gamma_{r-1}(a - b) \)
\( \frac{\Gamma_r a}{k - r + 1} > \Gamma_r(a - b) \)

Therefore, by mathematical induction, the condition for Step 3 is satisfied.
When $r = k$,

$$\Delta_{k+1} = \frac{\Gamma_{k-1}(a - b)}{2} + \frac{\Gamma_{k-1}(2b - a)}{2}$$

$$= \frac{\Gamma_{k-1}b}{2}$$

$$> 0$$

$$X_{2k-1}^* + X_{2k}^* = \frac{\Gamma_{k-1}(a + 2b)}{2(a + b)} + \frac{\Gamma_{k-1}a}{2(a + b)}$$

$$= \Gamma_{k-1}$$

Therefore, all the items are assigned.

Step 4 Case $r = k$

$$I_1 = I_2 - aX_{2k-1}^* + bX_{2k}^* + \Delta_k$$

$$= I_2 - \frac{\Gamma_{k-1}ba}{(a + b)} - \frac{\Delta_k a}{(a + b)} + \frac{\Gamma_{k-1}(a + b)}{ab} - \frac{\Delta_k b}{(a + b)} + \Delta_k$$

$$= I_2$$

Therefore, machine 2 can be kept busy.

$$\Delta_{k+1} = bX_{2k-1}^* - aX_{2k}^*$$

$$= \frac{\Gamma_{k-1}(b^2 - a^2)}{(a + b)} + \frac{\Delta_k (a + b)}{(a + b)}$$

$$= \Delta_k - \Gamma_{k-1}(a - b)$$

$$> 0$$

$$X_{2k-1}^* + X_{2k}^* = \frac{\Gamma_{k-1}b}{a + b} + \frac{\Gamma_{k-1}a}{a + b}$$

$$= \Gamma_{k-1}$$

Therefore, all the items are allocated.

Step 4 Case $r < k$

$$I_1 = I_2 - aX_{2r-1}^* + bX_{2r}^* + \Delta_r$$

$$= I_2 - \frac{\Delta_r(a + 2b)a}{2a(a + b)} - \frac{\Gamma_{r-1}(a - b)a}{2(a + b)} + \frac{\Delta_r b}{2(a + b)} - \frac{\Gamma_{r-1}(a - b)b}{2(a + b)} + \Delta_r$$

$$= I_2 - \frac{\Delta_r}{2} - \frac{\Gamma_{r-1}(a - b)}{2} + \Delta_r$$

$$= I_2 + \frac{\Delta_r}{2} - \frac{\Gamma_{r-1}(a - b)}{2}$$

$$> I_2$$
Therefore, \( I_2 = 0 \) is possible.

\[
X^*_{2r-1} + X^*_2 = \frac{\Delta_r(a + 2b)}{2a(a + b)} + \frac{\Delta_r}{2(a + b)}
\]

\[
= \frac{\Delta_r}{a}
\]

\[
\Gamma_r = \Gamma_{r-1} - X^*_2 - X^*_2
\]

\[
= \Gamma_{r-1} - \frac{\Delta_r}{a}
\]

\[
\Delta_{r+1} = bX^*_{2r-1} - aX^*_2
\]

\[
= \frac{\Delta_r(a + 2b)b}{2a(a + b)} + \frac{\Gamma_{r-1}(a - b)b}{2(a + b)} - \frac{\Delta_r a}{2(a + b)} + \frac{\Gamma_{r-1}(a - b)a}{2(a + b)}
\]

\[
= \frac{\Delta_r(ab + 2b^2 - a^2)}{2a(a + b)} + \frac{\Gamma_{r-1}(a - b)}{2}
\]

\[
= \frac{\Delta_r(2b - a)}{2a} + \frac{\Gamma_{r-1}(a - b)}{2}
\]

\[
= \frac{\Delta_r(2b - a)}{2a} + (\Gamma_r + \frac{\Delta_r}{a})(a - b)
\]

\[
= \frac{\Delta_r(2b - a) + \Gamma_r(a - b) + \Delta_r(a - b)}{2a}
\]

\[
= \frac{\Gamma_r(a - b) + \frac{\Delta_r b}{2a}}{2}
\]

\[
= \frac{\Gamma_r(a - b) + \frac{\Delta_r b}{2a} - \frac{\Gamma_r(a - b)}{2}}{2}
\]

But \( \frac{\Delta_r}{2} > \frac{\Gamma_r(a - b)}{2} \)

Therefore \( \Delta_{r+1} > \Gamma_r(a - b) \)

Therefore, if \( \Delta_r > \Gamma_{r-1}(a - b) \), then \( \Delta_{r+1} > \Gamma_r(a - b) \). Further, \( \Delta_1 > \Gamma_0(a - b) \). Hence, by mathematical induction, the condition for Case 2.2.3 will be satisfied by all remaining sublots and items.

Step 5 Case \( r = k \)

\[
I_1 = I_2 - aX^*_{2k-1} + bX^*_2 + \Delta_k
\]

\[
= I_2 - \frac{\Gamma_{k-1}a^2}{(a + b)} + \frac{\Gamma_{k-1}b^2}{(a + b)} + \Delta_k
\]
\[ = I_2 + \Delta_k - \Gamma_{k-1}(a - b) \]
\[ > I_2 \]

Therefore, machine 2 can be fully utilized.

\[ \Delta_{k+1} = bX_{2k-1}^* - aX_{2k}^* \]
\[ = \frac{\Gamma_{k-1}ab}{(a + b)} - \frac{\Gamma_{k-1}ba}{(a + b)} \]
\[ = 0 \]
\[ X_{2k-1}^* + X_{2k}^* = \frac{\Gamma_{k-1}a}{a + b} + \frac{\Gamma_{k-1}b}{a + b} \]
\[ = \Gamma_{k-1} \]

Therefore, all items are assigned.

Step 5 Case \( r < k \)

\[ I_1 = I_2 - aX_{2r-1}^* + bX_{2r}^* + \Delta_r \]
\[ = I_2 - \frac{\Gamma_{r-1}(2a^2 - b^2)a}{4a(a + b)} - \frac{\Gamma_{r-1}(2a - b)a}{4a} + \frac{\Gamma_{r-1}b^2}{4(a + b)} + \Delta_r \]
\[ = I_2 + \frac{\Gamma_{r-1}(3b^2 - 4a^2 - ab)}{4(a + b)} + \Delta_r \]
\[ = I_2 + \frac{\Gamma_{r-1}(a + b)(3b - 4a)}{4(a + b)} + \Delta_r \]
\[ = I_2 + \frac{\Gamma_{r-1}(3b - 4a)}{4} + \Delta_r \]
\[ = I_2 - \frac{(\Gamma_{r-1}a - \frac{3b}{4})}{\Delta_r} + \Delta_r \]

Since \( \Gamma_{r-1}a < \Delta_r \)
\[ \Delta_r > \Gamma_{r-1}(a - \frac{3b}{4}) \]

Therefore, \( I_1 > I_2 \)

Thus, \( I_2 = 0 \) is possible.

\[ X_{2r-1}^* + X_{2r}^* = \frac{\Gamma_{r-1}(2a^2 - b^2 + ab)}{4a(a + b)} + \frac{\Gamma_{r-1}(2a - b)}{4a} \]
\[ = \frac{\Gamma_{r-1}(2a - b)(a + b)}{4a(a + b)} + \frac{\Gamma_{r-1}(2a - b)}{4a} \]
\[ = \frac{\Gamma_{r-1}(2a - b)}{4a} + \frac{\Gamma_{r-1}(2a - b)}{4a} \]
Figure 4.6: A busy schedule for Case 2.2.3. \( a > b, s < t, Wa + s < Wb + t, n = 2k, C_{\text{max}} = s + Wa \).

\[
\begin{align*}
\Gamma_r &= \frac{\Gamma_{r-1}(2a - b)}{2a} \\
\Delta_{r+1} &= bX^{2r-1}_{2r-1} - aX^{2r}_{2r} \\
&= \frac{\Gamma_{r-1}(2a^2 - b^2)b}{4a(a + b)} + \frac{\Gamma_{r-1}(2a - b)b}{4a} \frac{\Gamma_{r-1}ab}{4(a + b)} \\
&= \frac{\Gamma_{r-1}(2a^2b - b^3) + (2a - b)(a + b)b - a^2b}{4a(a + b)} \\
&= \frac{\Gamma_{r-1}(3a^2b + ab^2 - 2b^3)}{4a(a + b)} \\
&= \frac{\Gamma_{r-1}b(3a^2 + ab - 2b^2)}{4a(a + b)} \\
&= \frac{\Gamma_{r-1}b(a + b)(3a - 2b)}{4a(a + b)} \\
&= \frac{\Gamma_{r-1}b(3a - 2b)}{4a} \\
&= \frac{\Gamma_{r-1}b(3a - 2b)}{2a} \frac{2}{2a} \\
&> \frac{\Gamma_{r-1}b(a - b)}{2a} \\
&> \frac{\Gamma_{r-1}(a - b)}{2a} + \frac{\Gamma_{r-1}b(a - b)}{2a} - \Gamma_{r-1}(a - b) \\
&> \frac{\Gamma_{r-1}(a - b)}{2a} + \frac{\Gamma_{r-1}(a - b)(b - 2a)}{2a} \\
&> \frac{\Gamma_{r-1} - \Gamma_{r-1}(2a - b)}{2a}) (a - b) \\
&> \Gamma_r(a - b)
\end{align*}
\]

Figure 4.6 depicts an optimal schedule.
Case 3 $a < b$.

Problems of Case 3 are the mirror image of the problems of Case 2. Therefore, optimal schedules are similar to those of Case 2.

### 4.3 Different Structures of Schedule

Different structures can be generated for a given product. These different structures play an important role in multi-product scheduling. We can generalize that there are three possible lot size structures:

1. Alternating pairs ($\rho_2$ schedule). An alternating pair schedule is one which has the following properties: there are $k$ time points $\tau_1, \ldots, \tau_k$ such that for $1 \leq i \leq k$, operations $aX_{2i-1}$ and $bX_{2i}$ complete at time $\tau_i$, as shown in Figure 4.7.

2. Forward flowshop. Here all sublots of a given product follow the sequence $M_1 \rightarrow M_2$, as shown in Figure 4.8.

3. Backward flowshop. Here all sublots of a given product follow the sequence $M_2 \rightarrow M_1$, as shown by considering the reverse schedule in Figure 4.8.

The above three structures can also be combined into a hybrid structure. Figure 4.9 shows a hybrid structure where sublots 1 and 7 are processed as forward flowshops, sublots
2, 3, 5, and 6 are processed as alternating pairs, and subplot 4 is processed as a backward flowshop.

4.4 Conclusion

We have shown that there exists a busy schedule for each case. Hence there is always an optimal schedule for the single product continuous-sized sublots. We have used the maximum number of sublots allowed or one less than that value if the maximum number of sublots is odd. However, optimal solutions can be obtained with any even number of sublots in the given number of maximum sublots. This is obvious by substituting $n$ by any even number which is less than $n$. Thus, single product continuous case can have multiple optimal solutions. Equal sublot size will not always give optimal schedule, but the equal-sized pairs give optimal schedules in most of the cases. The algorithms we provided for different cases can be solved in polynomial time. Further, these algorithms can be successfully used for integer case, when the number of items are very large and the number of sublots are few so that the effect of rounding off is negligible.
Chapter 5
Multiple Products

5.1 Introduction

Sriskandarajah and Wagneur (1999) explain how the schedule of a given lot can be divided into a head (H), a body (B) and a tail (T) for a no-wait flowshop problem. The head is the beginning portion of the schedule, in which machine $M_2$ (respectively, machine $M_1$) is idle throughout; the tail is the last portion of the schedule, where machine $M_1$ (resp., machine $M_2$) is idle throughout; the body is the middle portion of the schedule between the head and tail. In open shop problems, we can have a head or tail on either machine (see Profiles 1, 2, 3 and 4 in Figure 5.1).

In this chapter, we develop an algorithm to find all possible profiles. Then a cost matrix is generated and a TSP algorithm is used to get optimal product profile and product sequence. We consider the case without setup time. However, the algorithm can be successfully applied for the case with setup time with small modification.

Problem $F_2|\text{no - wait}|C_{\text{max}}$ can be formulated as a solvable case of the TSP with the cost matrix: $c_{j,j+1} = \max\{0, \beta_j - \alpha_{j+1}\}$, where job $j$ is followed by job $j+1$ in the production sequence, and $\alpha_j$ and $\beta_j$ are the processing times of job $j$ on machine 1 and machine 2, respectively. $c_{j,j+1}$ is the idle time on machine 1 after completion of job $j$ on machine 1 until the start of job $j+1$ on machine 1. $c_{j,j+1}$ can also be interpreted as the cost of traveling from city $j$ to city $j+1$ in TSP terminology. The algorithm of Gilmore and Gomory (1964) minimizes the makespan in TSP instances with the above cost matrix. We now show that, for given product profiles, an optimal product sequence for our problem can be obtained using the algorithm of Gilmore and Gomory (1964).
5.2 Scheduling and Algorithm Development

In this section, we analyze how product sequence can be obtained for given profile of each product, i.e., when the number and the sizes of sublots are known for each product. If the sublots sizes are given for products, Gilmore and Gomory algorithm gives the optimal product sequence. For the simultaneous problem of sublot sizes and product sequence, we develop a dynamic programming algorithm to get all the profiles and use a TSP algorithm to select the optimal profile of each product and product sequence.

5.2.1 Scheduling of Products with Given Profiles

When the profile of each product is known, i.e., the sublot sizes are already selected, we need to find the Optimal product sequence. The algorithm of Gilmore and Gomory (1964) provides optimal product sequence in this case.

**Theorem 2** For given product profiles, an optimal product sequence for the two machine no-wait openshop problem with lot streaming can be obtained using the algorithm of Gilmore and Gomory (1964).

Proof. In order to prove this result, we let heads \((\alpha_j)\) on machine \(M_1\) and tails \((\beta_j)\) on machine \(M_2\) as positive, and heads on machine \(M_2\) and tails on machine \(M_1\) as negative.
Then, we show that product sequencing problem in no-wait openshop can also be formulated as a TSP with the cost matrix \( c_{j,j+1} = \max\{0, \beta_j - \alpha_{j+1}\} \). We prove that this cost matrix is correct for all possible combinations of profiles, where product \( j+1 \) follows product \( j \).

If \( T_j = \beta_j \geq 0 \) and \( H_{j+1} = \alpha_{j+1} \geq 0 \), then product \( j \) has either profile 1 or 4 and product \( j+1 \) has either profile 1 or 3. The idle time on \( M_1 \) is \( c_{j,j+1} = \max\{0, T_j - H_{j+1}\} = \max\{0, \beta_j - \alpha_{j+1}\} \). If \( T_j = \beta_j \geq 0 \) and \( H_{j+1} = -\alpha_{j+1} > 0 \), then product \( j \) has either profile 1 or 4, and product \( j+1 \) has either profile 2 or 4. The idle time on \( M_1 \) is \( c_{j,j+1} = T_j + H_{j+1} = \beta_j - \alpha_{j+1} = \max\{0, \beta_j - \alpha_{j+1}\} \). If \( T_j = -\beta_j > 0 \) and \( H_{j+1} = \alpha_{j+1} \geq 0 \), then product \( j \) has either profile 2 or 3 and product \( j+1 \) has either profile 1 or 3. The idle time on \( M_1 \) is \( c_{j,j+1} = 0 = \max\{0, \beta_j - \alpha_{j+1}\} \). Finally, if \( T_j = -\beta_j > 0 \) and \( H_{j+1} = -\alpha_{j+1} > 0 \), then product \( j \) has either profile 2 or 3 and product \( j+1 \) has either profile 2 or 4. The idle time on \( M_1 \) is \( c_{j,j+1} = \max\{0, H_{j+1} - T_j\} = \max\{0, \beta_j - \alpha_{j+1}\} \).

Thus, the cost expression \( c_{j,j+1} = \max\{0, \beta_j - \alpha_{j+1}\} \) is valid for all profile combinations. Hence the algorithm of Gilmore and Gomory (1964) finds optimal product sequence in a two machine no-wait openshop for any given product profiles.  

### 5.2.2 Simultaneous Lot Streaming and Scheduling

When the profile is not given, then we have to find the optimal number of sublots, sublot sizes and the product sequence. We develop a dynamic programming algorithm and then incorporate it with a TSP algorithm to solve this difficult problem.

We let \( \lambda_h \) and \( \lambda_t \) are the length of head and tail, respectively of a given product profile. Note that we have to consider all possible profiles of each product for our scheduling problems with multiple products. First, the different profiles for each product must be found, as explained below.

Given a product profile for each of \( N \) products, let us consider a product sequence \( \sigma \) which is the order of product indices, i.e., products are scheduled in the order: \( P_\sigma(0), P_\sigma(1), P_\sigma(2), \ldots, P_\sigma(r-1), P_\sigma(r), P_\sigma(r+1), \ldots, P_\sigma(N), P_\sigma(N+1) \), as in Figure 5.2, where \( \sigma(i) \) denotes the index of the product that is scheduled in the \( i^{th} \) position of the sequence \( \sigma \). \( P_\sigma(0), P_\sigma(N+1) \)
are artificial products having zero length of head, body and tail. For convenient, we define slightly different cost matrix than the one described above, that is, distance from city (product) $\sigma(i)$ to city (product) $\sigma(i+1)$ is $d_{\sigma(i)\sigma(i+1)} + (B_{\sigma(i)} + B_{\sigma(i+1)})/2$ where $d_{\sigma(i)\sigma(i+1)}$ is the time between the start of tail of product $P_{\sigma(i)}$ and finish of head of product $P_{\sigma(i+1)}$ (see Figure 5.2). The reason for the choice becomes clear shortly.

Our problem is now simultaneous selection of profiles of products and scheduling. In order to get the optimal schedule, we introduce a TSP formulation. Each product is represented by a country, and each profile for each product is represented by a city in that country. Then our problem is: Starting and finishing at a given city in a given country, visit each country exactly once and visit exactly one city in each country. Let $B_{\sigma(i)}$ denote the cost to visit city $i_\sigma$ of country $i_\sigma$, and let $d_{i_\sigma,j_\sigma}$ denote the travel cost from city $i_\sigma$ to city $j_\sigma$. Since a TSP tour visits each city in each country, we need to adapt the TSP matrix to ensure that, on entering country $i$ at city $i_\sigma$, the tour next visits cities $i_{\sigma+1}, i_{\sigma+2}, \ldots, i_{\sigma+k_i-1(\text{mod}k_i)}$, where $k_i$ is the number of cities in country $i$, at zero cost. Note that $i_0 = i_k$. This is accomplished by specifying zero costs for that sequence and prohibitively large costs ($M^2$) for any other sequence within country $i$. Furthermore, since the tour will leave country $i$ from city $i_{\sigma+k_i-1(\text{mod}k_i)}$ when it enters that country at city $i_\sigma$, the cost of travel from city $i_{\sigma+k_i-1(\text{mod}k_i)}$ must be specified as $C_{i_{\sigma},j_\sigma}$. This is accomplished by setting $C_{i_{\sigma},j_\sigma} = d_{i_{\sigma+k_i-1(\text{mod}k_i)},j_\sigma} + [(B_{i_{\sigma+k_i-1(\text{mod}k_i)}} + B_{j_\sigma})/2]$, where the term in square brackets is half the total size of the bodies of $i_\sigma$ and $j_\sigma$. Finally, reentering a given country is prevented by the addition of a large cost ($M$) when traveling between any pair of countries. The
optimal cost, \( C \), will satisfy \( NM < C < (N + 1)M \).

**Example**

Consider two countries \( i \) and \( j \) with three and two cities respectively, as in Figure 5.3. We obtain the following TSP cost matrix.

\[
\begin{bmatrix}
  i_1 & i_2 & i_3 & j_1 & j_2 \\
i_1 & M^2 & 0 & M^2 & M + d_{i_2,j_1} + (B_{i_2} + B_{j_1})/2 \\
i_2 & M^2 & M^2 & 0 & M + d_{i_3,j_1} + (B_{i_3} + B_{j_1})/2 \\
i_3 & 0 & M^2 & M^2 & M + d_{i_1,j_1} + (B_{i_1} + B_{j_1})/2 \\
j_1 & M + d_{j_2,i_2} + (B_{j_2} + B_{i_2})/2 & M + d_{j_3,i_3} + (B_{j_3} + B_{i_3})/2 & M^2 & 0 \\
j_2 & M + d_{j_1,i_1} + (B_{j_1} + B_{i_1})/2 & M + d_{j_2,i_3} + (B_{j_2} + B_{i_3})/2 & M + d_{j_3,i_4} + (B_{j_3} + B_{i_4})/2 & 0 & M^2
\end{bmatrix}
\]

For example, if our solution follows the path \( i_1 \to j_2 \), then the corresponding TSP solution follows the path \( i_1 \to i_2 \to i_3 \to j_2 \to j_1 \), with total cost \( d_{i_1,j_1} + (B_{i_1} + B_{j_1})/2 \).

We now describe our overall algorithm, Global, for our problem. Let vector \( v = (v_1, v_2, v_3, v_4) \), where integers \( v_1 \) and \( v_2 \) (\( v_3 \) and \( v_4 \)) represent the sizes of first two (last two) sublots and \( v_k \geq 0 \), \( k = 1, 2, 3, 4 \). Since \( \mu_j \geq 2 \), we have \( W_j - 1 \geq v_1 + v_2 > 0 \) and \( W_j - 1 \geq v_3 + v_4 > 0 \). Given a vector \( v \) for a product such that \( v_1 + v_2 + v_3 + v_4 \leq W_j \),
the head and tail of the product is defined as \( \lambda_h = av_1 - bv_2, \lambda_t = bv_3 - av_4. \)

Algorithm Global

For product \( j, j = 1, \ldots, N: \)

For all \( v \) such that \( v_1 + v_2 + v_3 + v_4 \leq W_j: \)

Run \( DP^v(j) \) to find optimal integer subplot sizes corresponding to \( v. \)

Define city (i.e., product profile) \( k \) in country \( j \) by those optimal subplot sizes.

End For

End For

Set up the TSP cost matrix to enforce a visit to exactly one city in each country.

Run the TSP heuristic, output the solution and stop.

It remains to define \( DP^v(j) \), which is a dynamic programming algorithm. For each product \( j, \) this algorithm finds the optimal subplot sizes for a given vector \( v \) which defines \( \lambda_h \) and \( \lambda_t. \) It chooses the subplot sizes such that idle time on machine \( M_1 \) is minimized. By doing so, it generates all the possible profiles (i.e., cities) for all the products (i.e., countries), as required in our TSP formulation.

In order to describe our dynamic programming algorithm, we define a pair of integers \( Y_{2k-1}, Y_{2k} \geq 0, for k = 1, \ldots, n_j. \) We require that at least two and at most \( n_j \) \( Y_i \) values are positive integers. These positive \( Y_i \) values represent the optimal subplot sizes. Let \( N(S) \) denote the number of positive valued elements in the set \( S. \) In order to simplify the description and example that follow, we suppress \( v \) and \( j. \)

Algorithm \( DP^v(j) \)

Optimal Value Function

Let \( f_{2k}(Y_{2k-1}, Y_{2k}, w, r) \) denote the minimum idle time on \( M_1 \) for the partial schedule formed by sublots \( Y_{2k+1}, \ldots, Y_{2n_j}, \) given that the \((2k - 1)^{th}\) subplot has size \( Y_{2k-1} \) and the \(2k^{th}\) subplot has size \( Y_{2k}, \) and that there are \( w \) units and \( r \) sublots left to allocate.

[Comment: \( Y_{2k-1} \) and \( Y_{2k} \) are both nonzero if the schedule has alternating pair structure; \( Y_{2k-1} \) is nonzero and \( Y_{2k} \) is zero if the schedule has forward flowshop structure; \( Y_{2k-1} \) is zero and \( Y_{2k} \) is nonzero if the schedule has backward flowshop structure.]
**Boundary Conditions**

\[ f_{2t}(Y_{2t-1}, Y_{2t}, v_3 + v_4, r) = \max\{0, bY_{2t-1} + bv_4 - aY_{2t} - av_3\} \text{ for } N(\{v_3, v_4\}) \leq r \leq n_j - N(\{v_1, v_2\}), t \in \{1, \ldots, n_j - 2\}; f(\cdot, \cdot, \cdot, \cdot) = +\infty \text{ otherwise.} \]

**Optimal Solution**

\[ f_0(v_1, v_2, W_j - v_1 - v_2, n_j - N(\{v_1, v_2\})). \]

**Recurrence Relation**

\[ f_{2k}(Y_{2k-1}, Y_{2k}, w, r) = \min_{1 \leq v_{2k+1} + v_{2k+2} + Y_{2k-1} + Y_{2k} \leq w - (n_j - k) + 2}\{\max\{0, bY_{2k-1} + bY_{2k+2} - aY_{2k} - aY_{2k+1}\} + f_{2k+2}(Y_{2k+1}, Y_{2k+2}, w - Y_{2k+1} - Y_{2k+2}, r - N(\{Y_{2k+1}, Y_{2k+2}\}))\}. \]

It is important to note that three possible situations may occur at each stage:

i. \(Y_{2k-1} > 0\) and \(Y_{2k} = 0\), where the schedule follows the forward flowshop structure.

ii. \(Y_{2k-1} = 0\) and \(Y_{2k} > 0\), where the schedule follows the backward flowshop structure.

iii. \(Y_{2k-1} > 0\) and \(Y_{2k} > 0\), where the schedule follows the alternating pair structure.

The amount of computation time required by Algorithm Global can be calculated as follows. The number of possible values of \(j\) is \(O(N)\). The number of possible vectors \(v\) is \(O(W_j^4)\). The number of possible values of \(k\) is \(O(n_j)\). The number of possible values of \(w\) is \(O(W_j)\). Finally, the number of possible values of \(Y_{2k-1}\) and \(Y_{2k}\) in the recurrence relation is \(O(W_j^2)\). Thus, the overall computation time of Global is \(O(N \sum_{j=1}^{N} n_j W_j^7)\).

**Example:** Consider a single product \(j\), where \(W_j = 4\), \(n_j = 4\), \(a = 5\), and \(b = 4\). We now illustrate \(DP^v(j)\) using \(v = (1, 0, 0, 1)\), i.e. \(v_1 = 1\), \(v_2 = 0\), \(v_3 = 0\), \(v_4 = 1\), therefore \(\lambda_h = av_1 - bv_2 = 5\) and \(\lambda_i = bv_3 - av_4 = -5\).

The boundary conditions give:

\[ f_2(2, 0, 1, 2) = \max\{0, 4(2) + 4(1) - 5(0) - 5(0)\} = 12. \]

\[ f_2(0, 2, 1, 2) = \max\{0, 4(0) + 4(1) - 5(2) - 5(0)\} = 0. \]

\[ f_2(1, 1, 1, 1) = \max\{0, 4(1) + 4(1) - 5(1) - 5(0)\} = 3. \]

\[ f_4(1, 0, 1, 1) = \max\{0, 4(1) + 4(1) - 5(0) - 5(0)\} = 8. \]

\[ f_4(0, 1, 1, 1) = \max\{0, 4(0) + 4(1) - 5(1) - 5(0)\} = 0. \]
Then, using the recurrence relation at stage $k = 1$, we have:

$$f_2(1, 0, 2, 2) = \min \{\max \{0, 4(1) + 4(0) - 5(0) - 5(1)\} + f_4(1, 0, 1, 1), \max \{0, 4(1) + 4(1) - 5(0) - 5(0)\} + f_4(0, 1, 1, 1)\} = \min \{8, 8\},$$

where the choice is between $X_3 = 1, X_4 = 0$ and $X_3 = 0, X_4 = 1$, respectively.

$$f_2(0, 1, 2, 2) = \min \{\max \{0, 4(0) + 4(1) - 5(1) - 5(0)\} + f_4(1, 0, 1, 1), \max \{0, 4(1) + 4(1) - 5(1) - 5(0)\} + f_4(0, 1, 1, 1)\} = \min \{8, 0\},$$

where the choice is the same as above.

Similarly, using the recurrence relation at stage $k = 0$, we have:

$$f_0(1, 0, 3, 3) = \min \{\max \{0, 4(1) + 4(0) - 5(0) - 5(1)\} + f_2(1, 0, 2, 2), \max \{0, 4(1) + 4(0) - 5(0) - 5(2)\} + f_2(2, 0, 1, 2), \max \{0, 4(1) + 4(1) - 5(0) - 5(0)\} + f_2(0, 1, 2, 2), \max \{0, 4(1) + 4(2) - 5(0) - 5(0)\} + f_2(0, 2, 1, 2), \max \{0, 4(1) + 4(1) - 5(0) - 5(1)\} + f_2(1, 1, 1, 1)\} = \min \{8, 12, 8, 12, 6\},$$

where the choice is between $X_1 = 1, X_2 = 0$; $X_1 = 2, X_2 = 0$; $X_1 = 0, X_2 = 1$; $X_1 = 0, X_2 = 2$; and $X_1 = 1, X_2 = 1$, respectively.

There are seven possible solutions for the example, as we now discuss. In Figures 5.4 through 5.10, the makespan ($C_{\text{max}}$) equals the processing time on machine $M_1$ of $aW_j = 20$ units plus the idle time found by each schedule using $DP^*(j)$.

Solution 1: $f_0(1, 0, 3, 3) \to f_2(1, 0, 2, 2) \to f_4(1, 0, 1, 1)$. This gives a solution with $Y_1 = 1, Y_2 = 0, Y_3 = 1, Y_4 = 0$, as shown in Figure 5.4.

Solution 2: $f_0(1, 0, 3, 3) \to f_2(0, 1, 2, 2) \to f_4(1, 0, 1, 1)$. This gives a solution with $Y_1 = 0, Y_2 = 1, Y_3 = 1, Y_4 = 0$, as shown in Figure 5.5.

Solution 3: $f_0(1, 0, 3, 3) \to f_2(2, 0, 1, 2)$. This gives a solution with $Y_1 = 2, Y_2 = Y_3 = Y_4 = 0$, as shown in Figure 5.6.
Solution 4: $f_0(1, 0, 3, 3) \rightarrow f_2(1, 0, 2, 2) \rightarrow f_4(0, 1, 1, 1)$. This gives a solution with $Y_1 = 1, Y_2 = Y_3 = 0, Y_4 = 1$, as shown in Figure 5.7.

Solution 5: $f_0(1, 0, 3, 3) \rightarrow f_2(0, 1, 2, 2) \rightarrow f_4(0, 1, 1, 1)$. This gives a solution with $Y_1 = 0, Y_2 = 1, Y_3 = 0, Y_4 = 1$, as shown in Figure 5.8.

Solution 6: $f_0(1, 0, 3, 3) \rightarrow f_2(0, 2, 1, 2)$. This gives a solution with $Y_1 = 0, Y_2 = 2, Y_3 = Y_4 = 0$, as shown in Figure 5.9.
Solution 7: \( f_6(1, 0, 3, 3) \rightarrow f_2(1, 1, 1, 1) \). This gives an optimal solution with \( Y_1 = Y_2 = 1, Y_3 = Y_4 = 0 \), as shown in Figure 5.10.

We now prove that Algorithm \( DP^v(j) \) provides optimal subplot sizes for a given vector \( v \).

**Lemma 6** If \( X^* = (X_1^*, \ldots, X_n^*) \) is the vector of subplot sizes for a product given by \( DP^v(j) \)
for a given \( v \) (and thus given \( \lambda_h \) and \( \lambda_t \) values) and \( j \), and \( X = (X_1, \ldots, X_n) \) is any other vector of sublot sizes for the same \( \lambda_h \) and \( \lambda_t \), then we have \( B(X^*) \leq B(X) \), where \( B(X^*) \) is the length of body the product for sublot sizes \( X^* \).

Proof. Since \( DP^v(j) \) minimizes the cost over all possible state transitions, \( X^* \) minimizes the value of idle time \( (I_1) \) on \( M_1 \). Now, since \( \lambda_h + B(X^*) = I_1 + W_a \), where \( \lambda_h \) and \( W_a \) are constants, it follows that \( B(X^*) \leq B(X) \). \qed

**Theorem 3** The lot sizes \( X^*(j) = (X_{i,j}^*, \ldots, X_{n,j}^*) \) obtained by \( DP^v(j) \) minimize the makespan for given \( v \) (and thus given \( \lambda_h \) and \( \lambda_t \) values) and \( j \).

Proof: let \( \sigma \) be an optimal schedule of product indices, i.e., products are scheduled in the order: \( P_{\sigma(0)}, P_{\sigma(1)}, P_{\sigma(2)}, \ldots, P_{\sigma(r-1)} \), \( P_{\sigma(r)}, P_{\sigma(r+1)}, \ldots, P_{\sigma(N)}, P_{\sigma(N+1)} \), as in Figure 5.2. The makespan, \( F_{\sigma} \), can be written as \( F_{\sigma} = \hat{F}_{\sigma} + d_{\sigma(r-1)\sigma(r)} + d_{\sigma(r)\sigma(r+1)} + B_{\sigma(r)} \), where

\[
\hat{F}_{\sigma} = \sum_{i=1}^{r-1} d_{\sigma(i-1)\sigma(i)} + \sum_{i=r+2}^{N+1} d_{\sigma(i-1)\sigma(i)} + \sum_{i=1}^{r-1} B_{\sigma(i)} + \sum_{i=r+1}^{N+1} B_{\sigma(i)},
\]

where \( B_{\sigma(i)} \) is the body length of the product scheduled in the position \( \sigma(i) \).

Suppose there exists a product \( k \) scheduled in position \( \sigma(r) \) for which the sublot sizes are \( X(k) \neq X^*(k) \). We replace the lot sizes of \( X(k) \) with \( X^*(k) \) for this product. The makespan, \( F^*_{\sigma} \), can be written as

\[
F^*_{\sigma} = \hat{F}_{\sigma} + d_{\sigma(r-1)\sigma(r)} + d_{\sigma(r)\sigma(r+1)} + B_{\sigma(r)}.
\]

Since \( \lambda_h \) and \( \lambda_t \) are given, \( H_{\sigma(r)} = H_{\sigma(r)} \) and \( T_{\sigma(r)} = T_{\sigma(r)} \). Thus, \( d_{\sigma(r-1)\sigma(r)} = d_{\sigma(r-1)\sigma(r)} \) and \( d_{\sigma(r)\sigma(r+1)} = d_{\sigma(r)\sigma(r+1)} \). Also, from Lemma 6, \( B_{\sigma(r)} \leq B_{\sigma(r)} \). Thus, \( F^*_{\sigma} \leq F_{\sigma} \).

Similarly, we replace the lot sizes of \( X(j) \) with \( X^*(j) \) for each product \( j \) in the sequence \( \sigma \). Each such replacement will not increase the makespan. Thus, the sublot sizes \( X^*(j) = (X_{i,j}^*, \ldots, X_{n,j}^*) \), \( j = 1, 2, \ldots, N \) given by \( DP^v(j) \) minimize the makespan for the given \( v \) (i.e., \( \lambda_h \) and \( \lambda_t \) values). \qed

### 5.3 Conclusion

In this chapter, we have studied the problem of scheduling products with given sublot sizes as well as simultaneous lot streaming and scheduling when sublot sizes are not given. For the former case, we proved that the algorithm due to Gilmore and Gomory provides
an optimal schedule. Latter case is formulated as TSP for which there are many efficient solution techniques available. We have considered the problem without setup times. However, the problems with setup times can be solved by inserting the setup times with the first subplot pairs.
Chapter 6  
Computational Testing

In order to evaluate the performance of Algorithm Global, we test a large number of randomly generated two machine multi-product problem instances running on Sun computer. The algorithm GENIUS for TSP (Genreau, Hertz and Laporte, 1992) is received in Pascal and then translated into C language using the p2c translator. The algorithm Global is written in C language and the translated GENIUS is adopted for our problem.

For all problems tested, the processing times $a_j$ and $b_j$ are generated randomly from a uniform distribution over the interval $1 \leq a_j, b_j \leq 10$. When the number of products $N$ is in the range $2 \leq N \leq 8$, the number of sublots for each product $n_j$ is generated randomly from a uniform distribution over the interval $2 \leq n_j \leq 6$ and the total number of units for each product, $W_j$, is generated randomly from a uniform distribution over the interval $2 \leq W_j \leq 6$. When the number of products is in the range $9 \leq N \leq 13$, the number of sublots for each product $n_j$ is generated randomly from a uniform distribution over the interval $2 \leq n_j \leq 5$ and the total number of units for each product, $W_j$, is generated randomly from a uniform distribution over the interval $2 \leq W_j \leq 5$. When the number of products is in the range $14 \leq N \leq 20$, the number of sublots for each product $n_j$ is generated randomly from a uniform distribution over the interval $2 \leq n_j \leq 4$ and the total number of units for each product, $W_j$, is generated randomly from a uniform distribution over the interval $2 \leq W_j \leq 4$.

Each line in Table 6.1 represents results over ten randomly generated problems. The fourth column in Table 6.1 shows the mean CPU time required by Global.
<table>
<thead>
<tr>
<th>Data Set</th>
<th>$N$</th>
<th>$\max{W_j}$</th>
<th>Number Of Cities</th>
<th>CPU Time/(sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
<td>240</td>
<td>108.3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>325</td>
<td>158.7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
<td>480</td>
<td>287.6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>739</td>
<td>666.1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>720</td>
<td>621.1</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
<td>805</td>
<td>796.4</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>6</td>
<td>890</td>
<td>970.5</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>5</td>
<td>792</td>
<td>536.6</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>5</td>
<td>826</td>
<td>557.1</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>5</td>
<td>891</td>
<td>748.0</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>5</td>
<td>932</td>
<td>863.9</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>5</td>
<td>1016</td>
<td>1029.5</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>4</td>
<td>437</td>
<td>92.1</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>4</td>
<td>478</td>
<td>115.6</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>4</td>
<td>494</td>
<td>126.3</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>4</td>
<td>535</td>
<td>152.1</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>4</td>
<td>576</td>
<td>183.5</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>4</td>
<td>592</td>
<td>201.6</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>4</td>
<td>633</td>
<td>238.4</td>
</tr>
</tbody>
</table>

Table 6.1: Performance of Global.
Chapter 7

Some Related Results and Discussion

In this Chapter, we discuss some possible extensions of our algorithm Global. The algorithm Global can be utilized for single product integer-sized sublots, problems with attached setup times, sublots with pallet capacity, problems with transfer time between machines and two machine no-wait flowshops. We briefly discuss on these possibilities.

7.1 Single Product Integer-sized Sublots

In the single product case, first we can find all possible profiles using the dynamic programming algorithm. Then the cycle time can be calculated for each profile and the profile which gives the minimum cycle time is the optimal schedule.

7.2 Problems with Attached Setup Times

The algorithm Global can solve problems with attached setup time with a small modification. First dynamic programming algorithm can find the profiles without considering the setup times. The setup times only affect the head values. Let us call the head of profile \( k \) of product \( i \) after adjusting for the setup times as \( \text{adjusted head}_{i}^{k} \). Then

\[
(\text{adjusted head})_{i}^{k} = (\text{head})_{i}^{k} + s_{i} - t_{i},
\]

where \( (\text{head})_{i}^{k} \) is the head of profile \( k \) of product \( i \) before adjustment for setup times. Now these profiles adjusted for setup can be used in TSP to obtain the product sequence. The same adjustment can also be done for single product case.
7.3 Sublots with Restricted Capacity

We have assumed that sublots (pallets) have unlimited capacity. However, it is not true in all cases. The dynamic programming algorithm can be restricted to assign items to sublots within its capacity. This restriction reduces the number of possible profiles for a product. As a result the problem can be solved more easily than our original problem. Here we may assume that the pallets of a given product are identical.

7.4 Problems with Sublot Transfer Time

Generally, transferring sublots between machines consumes time. If this time is very large, we may need to reduce the number of sublots. Our model can consider such sublot transfer time with a small modification. In our explanation let $T_{12}$ be the transfer time from machine 1 to machine 2 and $T_{21}$ be the transfer time from machine 2 to machine 1. The Figures 7.1, 7.2 and 7.3 show how transfer time can be accommodated in a sublot pair. Since we use dynamic programming, transfer times can be simply included for each pair. Then the algorithm Global will solve for product sequence.

7.5 Two Machine No-Wait Flowshop

We have formally modeled the problem of simultaneous lot streaming and scheduling of multiple products in two machine no-wait openshops to minimize makespan. It has to be noted that the flowshop version of the problem is a special case of our openshop problem.
Figure 7.2: Backward flowshop structure with transfer time.

Figure 7.3: Alternating pair structure with transfer time.
since we need to consider only the forward flowshop during the construction of a schedule. Thus, the algorithm Global can easily be modified to solve two machine no-wait flowshops.

We now describe our overall algorithm, Global Flow, for flowshop problem. Let vector \( v = (v_1, v_4) \), where integer \( v_1 \) \( (v_4) \) represents the sizes of first (last) subplot and \( v_k \geq 0 \), \( k = 1 \) and \( k = 4 \). Since \( \mu_j \geq 2 \), we have \( W_j - 1 \geq v_1 > 0 \) and \( W_j - 1 \geq v_4 > 0 \). Given a vector \( v \) for a product such that \( v_1 + v_4 \leq W_j \), the head and tail of the product is defined as \( \lambda_h = av_1, \lambda_t = bv_4 \).

**Algorithm Global Flow**

For product \( j, j = 1, ..., N \):

For all \( v \) such that \( v_1 + v_4 \leq W_j \):

- Run \( DP^v(j) \) to find optimal integer subplot sizes corresponding to \( v \).

Define city (i.e., product profile) \( k \) in country \( j \) by those optimal subplot sizes.

End For

End For

Set up the TSP cost matrix to enforce a visit to exactly one city in each country.

Run the TSP heuristic, output the solution and stop.

It remains to define \( DP^v(j) \), which is a dynamic programming algorithm. For each product \( j \), this algorithm finds the optimal subplot sizes for a given vector \( v \) which defines \( \lambda_h \) and \( \lambda_t \). It chooses the subplot sizes such that idle time on machine \( M_1 \) is minimized. By doing so, it generates all the possible profiles (i.e., cities) for all the products (i.e., countries), as required in our TSP formulation.

In order to describe our dynamic programming algorithm, we define an integer \( Y_k \geq 0 \), for \( k = 1, ..., n_j \). The positive \( Y_i \) values represent the optimal subplot sizes.
Algorithm $DP^v(j)$

*Optimal Value Function*

Let $f_k(Y_k, w, r)$ denote the minimum idle time on $M_1$ for the partial schedule formed by sublots $Y_{k+1}, \ldots, Y_{n_j}$, given that the $(k-1)$th subplot has size $Y_{k-1}$ and that there are $w$ units and $r$ sublots left to allocate.

*Boundary Conditions*

$$f_t(Y_t, v, r) = \max\{0, bY_t - av_t\} \text{ for } 1 \leq r \leq n_j - 1, \ t \in \{1, \ldots, n_j - 1\}; \ f(\cdot, \cdot, \cdot, \cdot) = +\infty \text{ otherwise.}$$

*Optimal Solution*

$$f_0(v_1, W_j - v_1, n_j - 1).$$

*Recurrence Relation*

$$f_k(Y_k, w, r) = \min_{1 \leq Y_{k+1} \leq w}\{\max\{0, bY_k - aY_{k+1}\} + f_{k+1}(Y_{k+1}, w - Y_{k+1}, r - 1)\}.$$ 

The amount of computation time required by Algorithm Global Flow can be calculated as follows. The number of possible values of $j$ is $O(N)$. The number of possible vectors $v$ is $O(W_j^3)$. The number of possible values of $k$ is $O(n_j)$. The number of possible values of $w$ is $O(W_j)$. Finally, the number of possible values of $Y_k$ in the recurrence relation is $O(W_j)$. Thus, the overall computation time of Global Flow is $O\left(N \sum_{j=1}^{N} n_j W_j^4\right)$.

For the problems with pallet capacity, the dynamic programming will keep track of the items allocated to each subplot not to exceed the capacity. Thus, this will reduce the number of possible cities and hence the CPU time will be reduced.
Chapter 8

Conclusions

In this thesis we develop algorithms for minimizing the makespan in two machine no-wait open shops with lot streaming. For the single product continuous case we provide solution techniques which give optimal solution. We show that multi-product integer-sized sublot problem can be formulated as a traveling salesman problem with a pseudopolynomial number of cities. Many solution methods are available for solving large instances of the TSP efficiently (Lawler et al., 1985). Our computational tests of one such method indicate that it can routinely deliver close to optimal solutions to scheduling and lot streaming problem for instances with up to 20 products. The algorithm Global can also be used with some modification to solve for single product integer-sized sublot problem, problems with limited pallet capacity, problems with attached setup times and problems with transfer times. Further we have shown that the idea can be efficiently used for two machine no-wait flowshop problem. The problem studied here has the assumption of deterministic production environment. Possible future research could study the more general problems obtained by relaxing this assumption.
References


Appendices
Appendix 1: C Coded Program for the Dynamic Programming Algorithm

/* Appendix 1: C Coded Program for the Dynamic Programming Algorithm */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
int N;
int temp5, minw, minm, best_value;
int n, W, h, l1, l2, l3, i4, a, b, q, lambda1, lambdat;

typedef struct
{
    int head;
    int tail;
    int body;
    int X[20][2];
} Prof;

typedef struct
{
    int Number;
    Prof prof[4*4*4*4];
} Product;

typedef struct
{
    Product product[20];
} Global;

typedef struct
{
typedef struct
{
    int item;
    int sidle;
} Item;

typedef struct
{
    int slot;
    Item item[4];
} Lot;

typedef struct
{
    int seven;
    Lot lot[4];
} Even;

typedef struct
{
    int sodd;
    Even even[4];
} Odd;

typedef struct
{
    Odd odd[4];
} size;
typedef struct
{
    size sn[4];
} stage;

typedef struct
{
    int sn1;
    int sodd;
    int seven;
    int slot;
    int sitem;
    Next next;
} last;
FILE *outputf;
Global prod;

float Distance[1500][1500];
float sing = 1000000;
float squ = 1000;

void Final( stage s, int i, int odds, int evens,
    int lots, int items, int best_value)
{
    int tempodds, tempevens, tempj, templots, tempitems;
    int num, j;
    last last1;
    last next1;
    prod.product[h].prof[q-1].X[0][0]=i1;
    prod.product[h].prof[q-1].X[0][1]=i2;
    prod.product[h].prof[q-1].head=a*i1-b*i2;
    prod.product[h].prof[q-1].tail=b*i3-a*i4;
    num=a*i1-b*i2;
    num=(num>0)?num:0;
    j=b*i3+a*i4;
    j=(j>0)?j:0;
prod.product[h].prof[q-1].body=a*W+best_value-j-num;
num=0;

j=i;
do
{
    num=num+1;
    last1.sn1=j;
    last1.sodd=odds;
    last1.seven=evens;
    last1.sitem=items;
    last1.slot=lots;
    prod.product[h].prof[q-1].X[num][0]=odds;
    prod.product[h].prof[q-1].X[num][1]=evens;
    tempj = j;
    tempodds = odds;
    tempevens = evens;
    templots = lots;
    tempitems = items;
    j=s.sn[tempj-1].odd[odds].even[evens].lot[lots].item[items].next.sn1;
    odds=s.sn[tempj-1].odd[tempodds].even[tempevens].lot[templots].item[tempitems].next.sodd;
    evens=s.sn[tempj-1].odd[tempodds].even[tempevens].lot[templots].item[tempitems].next.seven;
    lots=s.sn[tempj-1].odd[tempodds].even[tempevens].lot[templots].item[tempitems].next.slot;
    items=s.sn[tempj-1].odd[tempodds].even[tempevens].lot[templots].item[tempitems].next.sitem;
}while(j!=n+1);
return;
}

stage secondlastassign( stage s,int odds,int evens,
int lots,int items,int r,int w)
{
    int j, temp6;
    int y;
    y=b*odds+b*i4-a*evens-a*i3;

    y=(y<0)?y:0;
    y=s.sn[n-1].odd[i3].even[i4].lot[lots+1].item[items+odds+evens].sdlle+y;
    s.sn[n-2].odd[odds].sodd=odds;
    s.sn[n-2].odd[odds].even[evens].seven=evens;
    s.sn[n-2].odd[odds].even[evens].lot[lots].slot=lots;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].sitem=items
;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].sidle=y;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].next.sn1=n;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].next.sodd=i3;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].next.seven=i4;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].next.slot=r;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].next.sitem=w;
s.sn[-2].odd[odds].even[evens].lot[lots].item[items].next.sidle=s.sn[n-1].odd[13].even[14].lot[r].item[w].sidle;

if (items == 0)
{
    temp6 = b*i1+b*evens-a*odds-a*i2;
temp6=(temp6>0)?temp6:0;
temp5 = y+temp6;
    if (temp5 < best_value)
    {
        best_value = temp5;
    }
}
return s;
}

stage assign( stage s,int i,int odds,int evens,int m,
int items,int e, int f,int best1,int temp,int 1)
{
int temp6;
s.sn[i-1].odd[odds].sodd=odds;
s.sn[i-1].odd[odds].even[evens].seven=evens;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].sitem=items;
s.sn[i-1].odd[odds].even[evens].lot[m].slot=m;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].sidle=temp;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].next.sodd=e;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].next.seven=f;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].next.sn1=i+1;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].next.slot=1;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].next.sitem=odds+evens+items;
s.sn[i-1].odd[odds].even[evens].lot[m].item[items].next.sidle=s.sn[i].odd[e].even[f].lot[l].item[odds+evens+items].sidle;
if (items == 0)
{
    temp6 = b*l1+b*evens-a*odds-a*i2;
    temp6=(temp6>0)?temp6:0;
    temp5 = temp++temp6;
    if (temp5 < best_value)
    {
        best_value = temp5;
    }
    Final(s,i, odds , evens ,m,items ,best_value);
}
}
return s;
}

void lastassign(stage s, int w, int i, int odds, int evens,int lots,int l)
{
    int e,f,e1,f1,y,temp,y1,beste,bestf,bestl,temp5,temp6;
    temp=500000;
    f=0;
    /* previous one follows forward flowshop structure */
    for(e=1;e<=w+1-odds-evens;e++)
    {
        if(minw <=e+f+odds+evens &&
        s.sn[i].odd[e].even[f].lot[l].item[odds+evens].sitem==odds+evens)
        {
            y=b*odds+b*f-a*evens-a*e;
            y=(y>0)?y:0;
            y=s.sn[i].odd[e].even[f].lot[l].item[odds+evens].sidle+y;
            if(y<temp)
            {
                temp =y;
                beste=e;
            }
bestf=f;
bestl=l;
}
}

/* previous one follows backward flowshop structure */

e1=0;
for(f1=1;f1<=w+1-odds;f1++)
{
if(minw <= e1+f1+odds+evens &&
s.sn[i].odd[e1].even[f1].lot[l].item[odds+evens].sitem==odds+evens)
{
y1=b*odds+b*f1-a*evens-a*e1;
y1=(y1>0)?y1:0;
y1=s.sn[i].odd[e1].even[f1].lot[l].item[odds+evens].sidle+y1;

if(y1<temp)
{
temp=y1;
beste=e1;
bestf=f1;
bestl=l;
}
}

temp6 = b*l1+b*evens-a*odds-a*l2;
temp6=(temp6>0)?temp6:0;
temp5 = temp+temp6;

if (temp5 < best_value)
{
    best_value = temp5;
    Final(s,i, odds , evens ,lots,0,best_value);
}
return;
}

main()
{
    int temp,beste,bestl,l,k,r,y,w,final, m, odds, evens;
    int num2,num3,num1,idle, total=0,items,lots;
    float tot_time;
    char M='M';
    stage s;
    int g,i,j,e,f,y1,e1,f1,num;
    void Final( stage s,int i, int odds ,int evens ,
    int lots, int items ,int best_value);
    stage secondlastassign(stage s,int odds,int evens, int lots ,int items,int r,int w);

    outputf = fopen("Output", "w");
    tot_time = clock()/(float)CLOCKS_PER_SEC;
    /*printf("\n No of Products = ");
    */
    scanf("%d", &N);
    /*printf(outputf, "\n No of Products = %d", N);*/

    for(h=0;h<N;h++)
    {
        q=0;
        /*printf(outputf, "\n\n------------------------------------------"roomId);
        fprintf(outputf, "\n\nProduct: %d",h+1);*/
        /*printf("\n Enter the number of sublots: ");
        */
        scanf("%d",&n);
        /*printf(outputf, "\n\n The number of sublots = %d", n);*/
        /*printf("\n Enter the number of items: ");
        */
        scanf("%d",&W);
        /*printf(outputf, "\n\n The number of items = %d", W);*/
        /*printf("\n Enter A: ");
        */
        scanf("%d",&a);
        }
/*printf(outputf, "\n\nProcessing Time on Machine 1 = \%d", a);*/
/*printf("\n Enter B:");
 */
scanf("\%d",&b);
/*printf(outputf, "\n\nProcessing Time on Machine 2 = \%d", b);*/
/*printf("\nProduct: \%d",h+1);*/

/*printf(outputf, "\n\nProduct: \%d",h+1);*/
for(i1=0;i1<=W-1;i1++)
{
    r=(i1==0)?1:0;
    for(i2=r;i2<=W-i1;i2++)
    {
        lambdas=a*i1-b*i2;
        for(i3=0;i3<=W-i1-i2;i3++)
        {
            r=(i3==0)?1:0;
            for(i4=r;i4<=W-i1-i2-i3;i4++)
            {
                lambdat=b*i3-a*i4;
                best_value = 500000000;
                final=3000;
                l=(i1>0&&i2>0)?2:1;
                k=(i3>0&&i4>0)?2:1;
                q=q+1;

                /***********

                Assigning the last pair, i.e., the n th pair

                ***********/

r=n-1-k;
w=W-i1-i2-i3-i4;
s.sn[n-1].odd[i3].sodd=l3;
s.sn[n-1].odd[i3].even[i4].seven=i4;
s.sn[n-1].odd[i3].even[i4].lot[r].slot=r;
s.sn[n-1].odd[i3].even[i4].lot[r].item[w].sitem=w;
\[ s.n-1].odd[i4].even[i4].lot[r].item[w].next_sn1=n+1; \\
y=b*13-a*14; \\
y=(y>0)?y:0; \\
y=s.sn[n-1].odd[i4].even[i4].lot[r].item[w].side+y; \\
if (w == 0) \\
{
    best\_value = b*11+b*14-a*13-a*12; 
    best\_value=(best\_value>0)?best\_value:0; 
    prod.product[h].prof[q-1].X[0][0]=i1; 
    prod.product[h].prof[q-1].X[0][1]=i2; 
    prod.product[h].prof[q-1].X[1][0]=i3; 
    prod.product[h].prof[q-1].X[1][1]=i4; 
    prod.product[h].prof[q-1].head=a*11-b*12; 
    prod.product[h].prof[q-1].tail=b*13-a*14; 
    num=a*11-b*12; 
    num=(num>0)?num:0; 
    j=-b*13+a*14; 
    j=(j>0)?j:0; 
    prod.product[h].prof[q-1].body=a\*W+best\_value-num-j; 
} \\

/*********************/

Assigning the second last pair, i.e., the (n-1)st pair

/*****************************/

/* Check whether it is the last possible pair */
/* If so, assign all the remaining items */
\[ m=r-1; \\
minm=m+1; \\
if (m==0) \\
{
    /* We can assign only forward or backward flowshop structure */
    s= secondlastassign(s,w,0,m,0,r,w); 
    s= secondlastassign(s,0,w,m,0,r,w); 
} \\
for(odds=1;odds<=w;odds++)
{ 
    evens=0;
    lots=m;
    items=w-odds;
    s=secondlastassign(s, odds, evens, lots, items, r, w);
}
for(evens=1; evens<=w; evens++)
{
    odds=0;
    lots=m;
    items=w-evens;
    s=secondlastassign(s, odds, evens, lots, items, r, w);
}

/* (n-1)st pair follows alternating structure */

/* alternating structure is possible if w>1 and m>0 */

/* if m=n, assign all the items */

if(w>1 && m==1)
{
    for(odds=1; odds<=w-1; odds++)
    {
        s=secondlastassign(s, odds, w-odds, m-1, 0, r, w);
    }
}
if(w>1 && m>1)
{
    for(odds=1; odds<=w-1; odds++)
    {
        for(evens=1; evens<=w-odds; evens++)
        {
            lots=m-1;
            items=w-odds-evens;
            s=secondlastassign(s, odds, evens, lots, items, r, w);
        }
    }
}
minw=w;

Assigning remaining pairs, i.e., (n-2),(n-3),...1

for(i=n-2;i>=1;i--)
{
    m=((m-1)>0)?m-1:0;
    w=w-1;
    l=m+1;
    minm=(minm>1)?minm-2:0;
    /* if it follows forward flowshop structure */
    evens=0;
    for(odds=minw;odds<=w;odds++)
    {
        for(lots=minm;lots<=m;lots++)
        {
            l=lots+1;
            if(lots==0)
            {
                lastassign(s,w,l,odds,evens,lots,l);
            }
        }
    }
    else
    {
        for(items=0;items<=w-odds;items++)
        {
            temp=500000;
            /* previous ones follows forward flowshop structure */
        
        f=0;
        for(e=1;e<=w+1-odds-items;e++)
        {
            if(minw<=e+f+odds+items && s.sn[i].odd[e].even[f].lot[1].item[odds+evens+items].site==odds+}
events+items)
{
    y=b*odds+b*f-a*evens-a*e;
    y=(y>0)?y:0;
    y=s.sn[i].odd[e].even[f].lot[l].item[odds+items+evens].sidle+y;
    if(y<temp)
    {
        temp=y;
        beste=e;
        bestf=f;
        bestl=l;
    }
}

/* previous one follows backward flowshop structure */

e1=0;
for(e1=1;e1<=w+1-odds-items;e1++)
{
    if(minw<=e1+f1+odds+items &&
       s.sn[i].odd[e1].even[f1].lot[l].item[odds+evens+items].sitem==odds+evens+items)
    {
        y=b*odds+b*f1-a*evens-a*e1;
        y=(y>0)?y:0;
        y=s.sn[i].odd[e1].even[f1].lot[l].item[odds+items+evens].sidle+y;
        if(y<temp)
        {
            temp=y;
            beste=e1;
            bestf=f1;
            bestl=l;
        }
    }
}

/* previous one follows alternating structure */
if((lots+2)<=r)
{
    for(e=1;e<=w+1-odds-items:e++)
    {
        for(f=1;f<=w+1-odds-items-e:f++)
        {
            if(minw<=e+f+odds+items &&
                s.sn[i].odd[e].even[f].lot[l].item[odds+evens+items].sitem==odds+evens+items)
            {
                y=b*odds+b*f-a*evens-a*e;
                y=(y>0)?y:0;
                y=s.sn[i].odd[e].even[f].lot[l].item[odds+items+evens].sidle+y;
                if(y<temp)
                {
                    temp=y;
                    beste=e;
                    bestf=f;
                    bestl=l;
                }
            }
        }
    }
}
if(temp!=500000)
{
    s=assign(s,l,odds,evens,lots,items,beste,bestf,bestl,temp,l);
}
/* if it follows backward flowshop structure */

odds=0;
l=m+1;
for(evens=minw;evens<=w;evens++)
{
    for(lots=minm;lots<=m;lots++)
    {
        l=lots+1;
        if(lots==0)
        {
            lastassign(s,w,i,odds,even,lots,l);
        }
        else
        {
            for(items=0;items<=w-evens;items++)
            {
                temp=500000;
            }
        }
    }
}

/* previous one follows forward flowshop structure */

f=0;
for(e=1;e<=w+1-evens-items;e++)
{
    if(minw<=e+f+evens+items &&
        s.sn[i].odd[e].even[f].lot[l].item[evens+items].sitem==evens+items)
    {
        y=b*odds+b*f-a*evens-a*e;
        y=(y>0)?y:0;
        y=s.sn[i].odd[e].even[f].lot[l].item[odds+items+evens].sidle+y;
        if(y<temp)
        {
            temp=y;
            beste=e;
            bestf=f;
            bestl=l;
        }
    }
}

/* previous one follows backward flowshop structure */
e1=0;
for(e1=1;e1<=w+1-evens-items;e1++)
{
    if(minw<=e1+f1+evens+items &&
        s.sn[i].odd[e1].even[f1].lot[l].item[odds+evens+items].sitem==odds+evens+items)
    {
        y=b*odds+b*f1-a*evens-a*e1;
        y=(y>0)?y:0;
        y=s.sn[i].odd[e1].even[f1].lot[l].item[odds+items+evens].sidle+y;
        if(y<temp)
        {
            temp=y;
            beste=e1;
            bestf=f1;
            bestl=l;
        }
    }
}

/* previous one follows alternating structure */

if((lots+2)<=r)
{
    for(e=1;e<=w+1-evens-items;e++)
    {
        for(f=1;f<=w+1-evens-items-e;f++)
        {
            if(minw<=e+f-evens+items &&
                s.sn[i].odd[e].even[f].lot[l].item[odds+evens+items].sitem==odds+
                evens+items)
            {
                y=b*odds+b*f-a*evens-a*e;
                y=(y>0)?y:0;
                y=s.sn[i].odd[e].even[f].lot[l].item[odds+items+evens].sidle+y;
                if(y<temp)
                {
                    temp=y;
                    beste=e;
        }
bestf=f;
bestl=l;
}
}
}
}
}

if (temp!=500000)
s=assign(s,l,odds,evens,lots,items,beste,bestf,bestl,temp,l);
}
}
}
}

/* if it follows alternating structure */
/* alternating structure is possible if w>1 and m>0 */

if(w>1 && m>0)
{
for(odds=minw;odds<=w-1;odds++)
{
for (evens=minw;evens<=w-odds;evens++)
{
for(lots=minm;lots<=m-1;lots++)
{
  l=lots+2;
  if(lots==1)
  lastassign(s,w,l,odds,evens,lots,l);
  else
  {
   for(items=0;items<=w-odds-evens;items++)
   {
    temp=500000;
    f=0;
    /* previous one follows forward flowshop structure */
for (e = 1; e <= w + 1 - odds - evens - items; e++)
{
    if (minw <= e + f + odds + evens + items &&
        s.sn[1].odd[e].even[f].lot[m + 1].item[odds + evens + items].sitem == evens + odds + items)
    {
        y = b * odds + b * f - a * evens - a * e;
        y = (y > 0) ? y : 0;
        y = s.sn[1].odd[e].even[f].lot[m + 1].item[evens + odds + items].sidle + y;
        if (y < temp)
        {
            temp = y;
            beste = e;
            bestf = f;
            bestl = m + 1;
        }
    }
}

/* previous one follows backward flowshop structure */

e1 = 0;
for (f1 = 1; f1 <= w + 1 - odds - evens - items; f1++)
{
    if (minw <= e1 + f1 + evens + odds + items &&
        s.sn[1].odd[e1].even[f1].lot[m + 1].item[evens + odds + items].sitem == evens + odds + items)
    {
        y1 = b * odds + b * f1 - a * evens - a * e1;
        y1 = (y1 > 0) ? y1 : 0;
        y1 = s.sn[1].odd[e1].even[f1].lot[m + 1].item[evens + odds + items].sidle + y1;
        if (y1 < temp)
        {
            temp = y1;
            beste = e1;
            bestf = f1;
            bestl = m + 1;
        }
    }
}
```c
if (temp < best_value)
    best_value = temp;
s=assign(s,i,odds,evens,m,0,beste,bestl,best,t,e,1);
}
}
}
}
}
}
minw=1;
if(m==0)
    break;
*/
/*printf("\n \n Profile: %d: ", q);
*/
/*fprintf(outputf, "\n \n Profile: %d:: ", q);*/
for(j=0;j<n;j++)
{
    /*printf("(%d , %d)",prod.product[h].prof[q-1].X[j][0],prod.product[h].prof[q-1].X[j][1]);
    */
    /*if(j<n-1)
        printf(".,");
    */
    /*fprintf(outputf, "(%d , %d)",prod.product[h].prod[q-1].X[j][0],prod.product[h].prof[q-1].X[j][1]);*/
    /*if(j<n-1)
        fprintf(outputf, ",");*/
}
/*printf(":Idle time=%d",best_value);
printf(":Head=%d",prod.product[h].prof[q-1].head);
printf(":Tail=%d",prod.product[h].prof[q-1].tail);
printf(":Body=%d",prod.product[h].prof[q-1].body);
*/
/*fprintf(outputf, "	 Idle time=%d",best_value);*/
}
CREATING DISTANCE MATRIX

num=0;
num1=0;
for(h=0;h<=N-1;h++)
{
    num=num1;
    num1=num+prod.product[h].Number;
    for(i=num;i<=num1-1;i++)
    {
        num2=0;
        num3=0;
        for(l=0;l<=N-1;l++)
        {
            num2=num3;
            num3=num2+prod.product[l].Number;
            for(j=num2;j<=num3-1;j++)
            {
                if(l==h)
                {
                    if(j==i+1 || (j==0 & & i==num1-1))
                    {
                        Distance[i][j]=0;
                    }
                }
            }
        }
    }
}
}
else
 {
 Distance[i][j]=sing;
 }
}
else
 {
 if (prod.product[h].prof[i-num].tail<=0 && prod.product[1].prof[j-num2].head<=0)
 {
 r=-prod.product[1].prof[j-num2].head;
 k=-prod.product[h].prof[i-num].tail;
 idle=(r>k)?r:k;
 }
 if (prod.product[h].prof[i-num].tail<=0 && prod.product[1].prof[j-num2].head>0)
 {
 }
 if (prod.product[h].prof[i-num].tail>0 && prod.product[1].prof[j-num2].head<=0)
 {
 }
 if (prod.product[h].prof[i-num].tail>0 && prod.product[1].prof[j-num2].head>0)
 {
 }
 if (i==num)
 {
 Distance[num-1-1][j]=squ+idle+(float) prod.product[h].prof[i-num].body/2+(float) prod.product[1].prof[j-num2].body/2;
 }
 }
else
 {
 Distance[-1-1][j]=squ+idle+(float) prod.product[h].prof[i-num].body/2+(float) prod.product[1].prof[j-num2].body/2;
 }
}
for(i=0;i<=total-1;i++)
{
    for(j=0;j<=total-1;j++)
    {
        printf("%.1f",Distance[i][j]);
    }
}
}
Appendix 2: Sample Output of a problem

We provide sample output of a problem with two products. For product 1, \( W = 4, n = 5, a = 1, b = 2 \) and for product 2, \( W = 4, n = 6, a = 2, b = 3 \). The Output "Profile" in Appendix 2.1, explains there are 85 profiles for product 1 and 155 profiles for product 2. Thus in our cost matrix, first 85, i.e., 1-85, cities (profiles) are of country 1 (product 1), and the remaining 155, i.e., 86-240, cities (profiles) are of country 2 (product 2).

The output "Path" in Appendix 2.2, explains that salesman has to leave product 1 at profile 24 and enter product 2 at profile 1. Therefore the selected profile for product 1 is 25, and for product 2 is 1.

Sublot sizes of profiles are given in the output "Profiles". It shows that for product 1, sublot sizes are \((0,3),(0,1),(0,1)\); for product 2, sublot sizes are \((0,1),(1,3),(0,1)\). Thus the schedule will be as follows:

\[
\begin{array}{cccccccc}
3 \times 1 & 2 \times 1 & 1 \times 2 & 1 \times 2 & 3 \times 2 & 1 \times 2 & 3 \times 1 & 2 \times 1 \\
3 \times 2 & 2 \times 2 & 1 \times 3 & 3 \times 3 & 1 \times 3 & 1 \times 3 & 3 \times 2 & 2 \times 2 \\
\end{array}
\]

cycle time = 32

Schedule for the sample output
Appendix 2.1: Output “Profile”

Data:

No of Products = 2

Product 1

The number of sublots = 4
The number of items = 5
Processing Time on Machine 1 = 1
Processing Time on Machine 2 = 2

Product 2

The number of sublots = 4
The number of items = 6
Processing Time on Machine 1 = 2
Processing Time on Machine 2 = 3

Product 1

Profile: 1:(0, 1),(1, 2),(0, 1),(0, 0)  Idle time=4
Profile: 2:(0, 1),(0, 2),(0, 2),(0, 0)  Idle time=5
Profile: 3:(0, 1),(0, 1),(0, 3),(0, 0)  Idle time=6
Profile: 4:(0, 1),(0, 4),(0, 0),(0, 0)  Idle time=7
Profile: 5:(0, 1),(1, 2),(1, 0),(0, 0)  Idle time=2
Profile: 6:(0, 1),(0, 2),(1, 1),(0, 0)  Idle time=3
Profile: 7:(0, 1),(0, 1),(1, 2),(0, 0)  Idle time=3
Profile: 8:(0, 1),(1, 3),(0, 0),(0, 0)  Idle time=4
Profile: 9:(0, 1),(1, 1),(2, 0),(0, 0)  Idle time=0
Profile: 10:(0, 1),(0, 1),(2, 1),(0, 0)  Idle time=1
Profile: 11:(0, 1),(2, 2),(0, 0),(0, 0)  Idle time=1
Profile: 12:(0, 1),(0, 1),(3, 0),(0, 0)  Idle time=1
Profile: 13:(0, 1),(3, 1),(0, 0),(0, 0)  Idle time=0
Profile: 14:(0, 1),(4, 0),(0, 0),(0, 0)  Idle time=0
Profile: 15:(0, 2),(0, 2),(0, 1),(0, 0)  Idle time=2
Profile: 16:(0, 2),(0, 1),(0, 2),(0, 0)  Idle time=3
Profile: 17:(0, 2),(0, 3),(0, 0),(0, 0)  Idle time=4
<table>
<thead>
<tr>
<th>Profile</th>
<th>Idle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>33</td>
<td>9</td>
</tr>
<tr>
<td>34</td>
<td>10</td>
</tr>
<tr>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>37</td>
<td>6</td>
</tr>
<tr>
<td>38</td>
<td>7</td>
</tr>
<tr>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>41</td>
<td>4</td>
</tr>
<tr>
<td>42</td>
<td>4</td>
</tr>
<tr>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>46</td>
<td>6</td>
</tr>
<tr>
<td>47</td>
<td>7</td>
</tr>
<tr>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td>49</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>54</td>
<td>3</td>
</tr>
<tr>
<td>55</td>
<td>4</td>
</tr>
</tbody>
</table>
Profile: 56: (1, 2), (1, 0), (1, 0), (0, 0)  Idle time=1
Profile: 57: (1, 2), (1, 1), (0, 0), (0, 0)  Idle time=1
Profile: 58: (1, 2), (2, 0), (0, 0), (0, 0)  Idle time=0
Profile: 59: (1, 3), (0, 1), (0, 0), (0, 0)  Idle time=1
Profile: 60: (1, 3), (1, 0), (0, 0), (0, 0)  Idle time=0
Profile: 61: (2, 0), (0, 2), (0, 1), (0, 0)  Idle time=8
Profile: 62: (2, 0), (0, 1), (0, 2), (0, 0)  Idle time=9
Profile: 63: (2, 0), (0, 3), (0, 0), (0, 0)  Idle time=10
Profile: 64: (2, 0), (1, 1), (1, 0), (0, 0)  Idle time=5
Profile: 65: (2, 0), (1, 0), (1, 1), (0, 0)  Idle time=6
Profile: 66: (2, 0), (1, 2), (0, 0), (0, 0)  Idle time=7
Profile: 67: (2, 0), (0, 1), (2, 0), (0, 0)  Idle time=6
Profile: 68: (2, 0), (2, 1), (0, 0), (0, 0)  Idle time=4
Profile: 69: (2, 0), (3, 0), (0, 0), (0, 0)  Idle time=1
Profile: 70: (2, 1), (1, 0), (0, 1), (0, 0)  Idle time=6
Profile: 71: (2, 1), (0, 2), (0, 0), (0, 0)  Idle time=7
Profile: 72: (2, 1), (1, 0), (1, 0), (0, 0)  Idle time=3
Profile: 73: (2, 1), (1, 1), (0, 0), (0, 0)  Idle time=4
Profile: 74: (2, 1), (2, 0), (0, 0), (0, 0)  Idle time=1
Profile: 75: (2, 2), (0, 1), (0, 0), (0, 0)  Idle time=4
Profile: 76: (2, 2), (1, 0), (0, 0), (0, 0)  Idle time=1
Profile: 77: (3, 0), (0, 1), (0, 1), (0, 0)  Idle time=9
Profile: 78: (3, 0), (0, 2), (0, 0), (0, 0)  Idle time=10
Profile: 79: (3, 0), (0, 1), (1, 0), (0, 0)  Idle time=8
Profile: 80: (3, 0), (1, 1), (0, 0), (0, 0)  Idle time=7
Profile: 81: (3, 0), (2, 0), (0, 0), (0, 0)  Idle time=4
Profile: 82: (3, 1), (0, 1), (0, 0), (0, 0)  Idle time=7
Profile: 83: (3, 1), (1, 0), (0, 0), (0, 0)  Idle time=4
Profile: 84: (4, 0), (0, 1), (0, 0), (0, 0)  Idle time=10
Profile: 85: (4, 0), (1, 0), (0, 0), (0, 0)  Idle time=7

Product 2

Profile: 86: (0, 1), (1, 3), (0, 1), (0, 0)  Idle time=5
Profile: 87: (0, 1), (0, 3), (0, 2), (0, 0)  Idle time=7
Profile: 88: (0, 1), (0, 2), (0, 3), (0, 0)  Idle time=9
Profile: 89: (0, 1), (0, 1), (0, 4), (0, 0)  Idle time=11
<table>
<thead>
<tr>
<th>Profile</th>
<th>Idle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>13</td>
</tr>
<tr>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>7</td>
</tr>
<tr>
<td>93</td>
<td>4</td>
</tr>
<tr>
<td>94</td>
<td>6</td>
</tr>
<tr>
<td>95</td>
<td>8</td>
</tr>
<tr>
<td>96</td>
<td>0</td>
</tr>
<tr>
<td>97</td>
<td>4</td>
</tr>
<tr>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>104</td>
<td>1</td>
</tr>
<tr>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>106</td>
<td>2</td>
</tr>
<tr>
<td>107</td>
<td>4</td>
</tr>
<tr>
<td>108</td>
<td>3</td>
</tr>
<tr>
<td>109</td>
<td>8</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>2</td>
</tr>
<tr>
<td>112</td>
<td>2</td>
</tr>
<tr>
<td>113</td>
<td>3</td>
</tr>
<tr>
<td>114</td>
<td>0</td>
</tr>
<tr>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>0</td>
</tr>
<tr>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>118</td>
<td>0</td>
</tr>
<tr>
<td>119</td>
<td>0</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>121</td>
<td>4</td>
</tr>
<tr>
<td>122</td>
<td>3</td>
</tr>
<tr>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>0</td>
</tr>
<tr>
<td>125</td>
<td>0</td>
</tr>
<tr>
<td>126</td>
<td>0</td>
</tr>
<tr>
<td>127</td>
<td>0</td>
</tr>
</tbody>
</table>
Profile: 128:(0, 3), (3, 0), (0, 0), (0, 0) Idle time=0
Profile: 129:(0, 4), (0, 1), (0, 1), (0, 0) Idle time=1
Profile: 130:(0, 4), (0, 2), (0, 0), (0, 0) Idle time=0
Profile: 131:(0, 4), (0, 1), (1, 0), (0, 0) Idle time=0
Profile: 132:(0, 4), (1, 1), (0, 0), (0, 0) Idle time=0
Profile: 133:(0, 4), (2, 0), (0, 0), (0, 0) Idle time=0
Profile: 134:(0, 5), (0, 1), (0, 0), (0, 0) Idle time=0
Profile: 135:(0, 5), (1, 0), (0, 0), (0, 0) Idle time=0
Profile: 136:(1, 0), (1, 3), (0, 1), (0, 0) Idle time=10
Profile: 137:(1, 0), (0, 3), (0, 2), (0, 0) Idle time=12
Profile: 138:(1, 0), (0, 2), (0, 3), (0, 0) Idle time=14
Profile: 139:(1, 0), (0, 1), (0, 4), (0, 0) Idle time=16
Profile: 140:(1, 0), (0, 5), (0, 0), (0, 0) Idle time=18
Profile: 141:(1, 0), (2, 2), (1, 0), (0, 0) Idle time=5
Profile: 142:(1, 0), (3, 0), (1, 1), (0, 0) Idle time=10
Profile: 143:(1, 0), (0, 2), (1, 2), (0, 0) Idle time=9
Profile: 144:(1, 0), (1, 0), (1, 3), (0, 0) Idle time=11
Profile: 145:(1, 0), (1, 4), (0, 0), (0, 0) Idle time=13
Profile: 146:(1, 0), (2, 1), (2, 0), (0, 0) Idle time=2
Profile: 147:(1, 0), (2, 0), (2, 1), (0, 0) Idle time=5
Profile: 148:(1, 0), (1, 0), (2, 2), (0, 0) Idle time=6
Profile: 149:(1, 0), (2, 3), (0, 0), (0, 0) Idle time=8
Profile: 150:(1, 0), (1, 1), (3, 0), (0, 0) Idle time=4
Profile: 151:(1, 0), (1, 0), (3, 1), (0, 0) Idle time=1
Profile: 152:(1, 0), (3, 2), (0, 0), (0, 0) Idle time=3
Profile: 153:(1, 0), (0, 1), (4, 0), (0, 0) Idle time=6
Profile: 154:(1, 0), (4, 1), (0, 0), (0, 0) Idle time=0
Profile: 155:(1, 0), (5, 0), (0, 0), (0, 0) Idle time=0
Profile: 156:(1, 1), (0, 3), (0, 1), (0, 0) Idle time=10
Profile: 157:(1, 1), (0, 2), (0, 2), (0, 0) Idle time=9
Profile: 158:(1, 1), (0, 1), (0, 3), (0, 0) Idle time=11
Profile: 159:(1, 1), (0, 4), (0, 0), (0, 0) Idle time=13
Profile: 160:(1, 1), (3, 0), (1, 0), (0, 0) Idle time=7
Profile: 161:(1, 1), (0, 2), (1, 1), (0, 0) Idle time=7
Profile: 162:(1, 1), (0, 1), (1, 2), (0, 0) Idle time=6
Profile: 163:(1, 1), (1, 3), (0, 0), (0, 0) Idle time=8
Profile: 164:(1, 1), (2, 0), (2, 0), (0, 0) Idle time=2
Profile: 165:(1, 1), (0, 1), (2, 1), (0, 0) Idle time=4
Profile: 166:(1, 1),(2, 2),(0, 0),(0, 0)  Idle time=3
Profile: 167:(1, 1),(1, 0),(3, 0),(0, 0)  Idle time=0
Profile: 168:(1, 1),(3, 1),(0, 0),(0, 0)  Idle time=0
Profile: 169:(1, 1),(4, 0),(0, 0),(0, 0)  Idle time=0
Profile: 170:(1, 2),(0, 2),(0, 1),(0, 0)  Idle time=5
Profile: 171:(1, 2),(0, 1),(0, 2),(0, 0)  Idle time=6
Profile: 172:(1, 2),(0, 3),(0, 0),(0, 0)  Idle time=8
Profile: 173:(1, 2),(2, 0),(1, 0),(0, 0)  Idle time=4
Profile: 174:(1, 2),(0, 1),(1, 1),(0, 0)  Idle time=2
Profile: 175:(1, 2),(1, 2),(0, 0),(0, 0)  Idle time=3
Profile: 176:(1, 2),(1, 0),(2, 0),(0, 0)  Idle time=0
Profile: 177:(1, 2),(2, 1),(0, 0),(0, 0)  Idle time=0
Profile: 178:(1, 2),(3, 0),(0, 0),(0, 0)  Idle time=0
Profile: 179:(1, 3),(0, 1),(0, 1),(0, 0)  Idle time=1
Profile: 180:(1, 3),(0, 2),(0, 0),(0, 0)  Idle time=3
Profile: 181:(1, 3),(0, 1),(1, 0),(0, 0)  Idle time=0
Profile: 182:(1, 3),(1, 1),(0, 0),(0, 0)  Idle time=0
Profile: 183:(1, 3),(2, 0),(0, 0),(0, 0)  Idle time=0
Profile: 184:(1, 4),(0, 1),(0, 0),(0, 0)  Idle time=0
Profile: 185:(1, 4),(1, 0),(0, 0),(0, 0)  Idle time=0
Profile: 186:(2, 0),(1, 2),(0, 1),(0, 0)  Idle time=12
Profile: 187:(2, 0),(0, 2),(0, 2),(0, 0)  Idle time=14
Profile: 188:(2, 0),(0, 1),(0, 3),(0, 0)  Idle time=16
Profile: 189:(2, 0),(0, 4),(0, 0),(0, 0)  Idle time=18
Profile: 190:(2, 0),(2, 1),(1, 0),(0, 0)  Idle time=7
Profile: 191:(2, 0),(2, 0),(1, 1),(0, 0)  Idle time=9
Profile: 192:(2, 0),(1, 0),(1, 2),(0, 0)  Idle time=11
Profile: 193:(2, 0),(1, 3),(0, 0),(0, 0)  Idle time=13
Profile: 194:(2, 0),(1, 1),(2, 0),(0, 0)  Idle time=7
Profile: 195:(2, 0),(1, 0),(2, 1),(0, 0)  Idle time=6
Profile: 196:(2, 0),(2, 2),(0, 0),(0, 0)  Idle time=8
Profile: 197:(2, 0),(0, 1),(3, 0),(0, 0)  Idle time=9
Profile: 198:(2, 0),(3, 1),(0, 0),(0, 0)  Idle time=3
Profile: 199:(2, 0),(4, 0),(0, 0),(0, 0)  Idle time=0
Profile: 200:(2, 1),(2, 0),(0, 1),(0, 0)  Idle time=9
Profile: 201:(2, 1),(1, 0),(0, 2),(0, 0)  Idle time=11
Profile: 202:(2, 1),(0, 3),(0, 0),(0, 0)  Idle time=13
Profile: 203:(2, 1),(2, 0),(1, 0),(0, 0)  Idle time=4
Profile: 204:(2, 1),(0, 1),(1, 1),(0, 0)  Idle time=7
Profile: 205:(2, 1),(1, 2),(0, 0),(0, 0)  Idle time=8
Profile: 206:(2, 1),(1, 0),(2, 0),(0, 0)  Idle time=2
Profile: 207:(2, 1),(2, 1),(0, 0),(0, 0)  Idle time=3
Profile: 208:(2, 1),(3, 0),(0, 0),(0, 0)  Idle time=0
Profile: 209:(2, 2),(1, 0),(0, 1),(0, 0)  Idle time=6
Profile: 210:(2, 2),(0, 2),(0, 0),(0, 0)  Idle time=8
Profile: 211:(2, 2),(1, 0),(1, 0),(0, 0)  Idle time=1
Profile: 212:(2, 2),(1, 1),(0, 0),(0, 0)  Idle time=3
Profile: 213:(2, 2),(2, 0),(0, 0),(0, 0)  Idle time=0
Profile: 214:(2, 3),(0, 1),(0, 0),(0, 0)  Idle time=3
Profile: 215:(2, 3),(1, 0),(0, 0),(0, 0)  Idle time=0
Profile: 216:(3, 0),(1, 1),(0, 1),(0, 0)  Idle time=14
Profile: 217:(3, 0),(0, 1),(0, 2),(0, 0)  Idle time=16
Profile: 218:(3, 0),(0, 3),(0, 0),(0, 0)  Idle time=18
Profile: 219:(3, 0),(1, 1),(1, 0),(0, 0)  Idle time=10
Profile: 220:(3, 0),(1, 0),(1, 0),(0, 0)  Idle time=11
Profile: 221:(3, 0),(1, 2),(0, 0),(0, 0)  Idle time=13
Profile: 222:(3, 0),(0, 1),(2, 0),(0, 0)  Idle time=12
Profile: 223:(3, 0),(2, 1),(0, 0),(0, 0)  Idle time=8
Profile: 224:(3, 0),(3, 0),(0, 0),(0, 0)  Idle time=3
Profile: 225:(3, 1),(1, 0),(0, 1),(0, 0)  Idle time=11
Profile: 226:(3, 1),(0, 2),(0, 0),(0, 0)  Idle time=13
Profile: 227:(3, 1),(1, 0),(1, 0),(0, 0)  Idle time=6
Profile: 228:(3, 1),(1, 1),(0, 0),(0, 0)  Idle time=8
Profile: 229:(3, 1),(2, 0),(0, 0),(0, 0)  Idle time=3
Profile: 230:(3, 2),(0, 1),(0, 0),(0, 0)  Idle time=8
Profile: 231:(3, 2),(1, 0),(0, 0),(0, 0)  Idle time=3
Profile: 232:(4, 0),(0, 1),(0, 1),(0, 0)  Idle time=16
Profile: 233:(4, 0),(0, 1),(0, 0),(0, 0)  Idle time=18
Profile: 234:(4, 0),(0, 1),(1, 0),(0, 0)  Idle time=15
Profile: 235:(4, 0),(1, 1),(0, 0),(0, 0)  Idle time=13
Profile: 236:(4, 0),(2, 0),(0, 0),(0, 0)  Idle time=8
Profile: 237:(4, 1),(0, 1),(0, 0),(0, 0)  Idle time=13
Profile: 238:(4, 1),(1, 0),(0, 0),(0, 0)  Idle time=8
Profile: 239:(5, 0),(0, 1),(0, 0),(0, 0)  Idle time=18
Profile: 240:(5, 0),(1, 0),(0, 0),(0, 0)  Idle time=13
Appendix 2.2: Output "Path"

→ 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 →
14 → 15 → 16 → 17 → 18 → 19 → 20 → 21 → 22 → 23 → 24 → 86 → 87
→ 88 → 89 → 90 → 91 → 92 → 93 → 94 → 95 → 96 → 97 → 98 → 99 →
100 → 101 → 102 → 103 → 104 → 105 → 106 → 107 → 108 → 109 → 110 → 111 →
112 → 113 → 114 → 115 → 116 → 117 → 118 → 119 → 120 → 121 → 122 → 123 →
124 → 125 → 126 → 127 → 128 → 129 → 130 → 131 → 132 → 133 → 134 → 135 →
136 → 137 → 138 → 139 → 140 → 141 → 142 → 143 → 144 → 145 → 146 → 147 → 148 →
149 → 150 → 151 → 152 → 153 → 154 → 155 → 156 → 157 → 158 → 159 → 160 → 161 → 162 →
163 → 164 → 165 → 166 → 167 → 168 → 169 → 170 → 171 → 172 → 173 → 174 → 175 → 176 →
177 → 178 → 179 → 180 → 181 → 182 → 183 → 184 → 185 → 186 → 187 → 188 → 189 → 190 → 191 →
192 → 193 → 194 → 195 → 196 → 197 → 198 → 199 → 200 → 201 → 202 → 203 → 204 → 205 →
206 → 207 → 208 → 209 → 210 → 211 → 212 → 213 → 214 → 215 → 216 → 217 → 218 → 219 → 220 →
221 → 222 → 223 → 224 → 225 → 226 → 227 → 228 → 229 → 230 → 231 → 232 → 233 → 234 →
235 → 236 → 237 → 238 → 239 → 240 → 241 → 242 → 243 → 244 → 245 → 246 → 247 → 248 → 249 → 250 → 251 →
252 → 253 → 254 → 255 → 256 → 257 → 258 → 259 → 260 → 261 → 262 → 263 → 264 → 265 → 266 → 267 → 268 → 269 →
270 → 271 → 272 → 273 → 274 → 275 → 276 → 277 → 278 → 279 → 280 → 281 → 282 → 283 → 284 → 285

The Total Processing time is 108.250000 Secs.