BATCH SCHEDULING IN SUPPLY CHAINS

BATCH SCHEDULING IN SUPPLY CHAINS

By
Esaignani Selvarajah, B.Sc.(Eng.), M.A.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for Ph.D.

McMaster University

Ph.D in M/IS (2006)

McMaster University Hamilton Ontario

TITLE: Batch Scheduling in Supply Chains

AUTHOR: Esaignani Selvarajah
B.Sc. (University of Peradeniya)
M.A.Sc. (University of Toronto)

SUPERVISOR: Professor George Steiner

Abstract

Supply chain management is a major issue in many industries as firms realize the importance of creating an integrated relationship with their suppliers and customers. In many manufacturing organizations minimizing the total cost of inventory holding and delivery plays a major role in production scheduling. Inventory holding cost is proportional to the flow time of jobs at the shop. Therefore, we study single machine batch scheduling problems to minimize the sum of weighted flow time and the delivery cost in supply chains.

It has been proven that many single machine batch scheduling problems even at the supplier level and the manufacturer level are hard problems to be solved. Therefore, batch scheduling problems for supplier-manufacturer coordination are even harder. Hence, heuristic algorithms may be developed to solve such problems. A good heuristic can be developed only when the specific properties of the given problem are analyzed thoroughly. Since there are many problems at the supplier level and manufacturer level not yet solved, we study single machine scheduling problems under different conditions at the supplier and manufacturer. Then we study batch scheduling problems in a supplier-manufacturer systerm.

We first study some polynomially solvable problems at the supplier and at the manufacturer. Batch scheduling problems at the supplier when jobs have aribtrary processing times and arbitrary weights are intractable. We provide a 2-approximation algorithm for this problem. The performance of this 2-approximation algorithm shows that it provides close to optimal solutions for practical situations. Batch scheduling problems at the manufacturer of multi-product case is intractable even if the weights are identical. We provide a 2-approximation algorithm for this problem and a hybrid meta-heuristic algorithm for the arbitrary weight case. We develop an algorithm for the lower bound of this problem and compare the result of the heuristic algorithm with that of the lower bound solution.

Then some batch scheduling problems at the manufacturer in a customer centric supply chain are analysed and dynamic programming algorithms are developed to solve these problems optimally. Finally batch scheduling with supplier manufacturer coordination is studied and there again dynamic programming algorithms are developed to solve the batch scheduling problems of given job sequence under two different conditions.

Acknowledgements

I would like to express my sincere gratitude to Professor George Steiner who helped me profusely at various stages of this thesis work. His constant guidance and profound involvement in my research need appreciation. His extended financial support, encouragement, and patience cannot be rewarded by the simple word "thank you".

I would like to thank Professor Prakash Abad, and Professor Brian Baetz for their valuable comments and suggestions which improved the content and the presentation of my thesis. I would like to thank my external examiner Professor Fazle Baki for providing valuable comments on my thesis. I am also thankful to Professor Mahmut Parlar for his encouragement and various supports throughout my studies. I would like to thank Professor Elkafi Hassini for his valuable information and advices. I cannot forget my great times with Tiina Salisbury when I was the teaching assistant for her courses.

I thank McMaster University, and Natural Science and Engineering Research Council of Canada for their financial support throughout my studies. I thank all my colleagues who made my stay at McMaster enjoyable. The administrative assistance provided by departmental staff Linda Kszytan, Carolyn Colwell, Vicki Cometto, and Sandra Stephen and computer system administrators Steve Bendo, David Ryan, and Carver Lewis is deeply remembered.

I express my deep gratitude to my parents who are always a source of motivation and inspiration to me.

Dedication

I dedicate this thesis to

my beloved grandfather late Mr. Sinnaiya Ponnappah and

my beloved grandmother late Mrs. Kanmani Ponnappah

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Sequencing and Scheduling

Sequencing and scheduling decisions play a crucial role in manufacturing and service industries, and information processing environments. Scheduling involves allocation of limited resources such as equipment, labor and space to jobs, activities, tasks, or customers through time. A sequencing problem involves finding a sequence in which to process a set of tasks, that minimizes a given cost function. Generally a scheduling problem involves determining a detailed assignment of jobs to machines over a period of time, that minimizes a given cost function.

We use the terms product, item, and job throughout the thesis. The term *product* represents the product type which is manufactured. Generally multiple copies of the same product type are manufactured in a manufacturing environment. We call the single unit of that product type *item*. When we study manufacturing environments where multiple product types are manufactured, we call a single unit of any product type a *job*. In machine scheduling terminology, *processing time* of a job is the time required to process it in the shop, and *release time* of a job is the time at which the job arrives in the shop from its immediate supplier.

1.2 Machine Scheduling

There are three levels of decisions in any business enterprise such as strategic decision, tactical decision, and operational decision. In a manufacturing enterprise, on the tactical level medium term decisions are made, such as weekly demand forecasts, distribution and transportation planning, production planning, and material requirements planning. The operational level is concerned with the very short term decisions made from day to day. The border between the tactical and operational levels is vague. Production planning problems are generally called *machine scheduling* or production scheduling.

Machine scheduling includes worker and machine assignment, job sequencing, and the coordination of material handling and maintenance support. In a competitive business environment, efficient and effective machine scheduling has become a necessity for survival in the market. Therefore, machine scheduling has attracted many researchers since the early 1950s and an impressive amount of literature has been created. Broadly speaking machine scheduling is the translation of customer orders into production schedules. Even though machine scheduling arose originally in a manufacturing context, various other applications are also possible. Jobs and machines can stand for patients and hospital equipment, runways and take-offs and landings at an airport, classes and teachers, programs and computer processors, cities and traveling salesman, and/or projects and payments.

1.2.1 Machine Scheduling Models

The major machine scheduling models are categorized by specifying the resource configuration and the nature of the tasks. For example, a model may contain one machine or several machines; the set of jobs available for scheduling may not change over time (also called static system) or new jobs appear over time (also called dynamic system); machines or buffers may have limited capacity; there may be technological restrictions on the job processing order, or no-wait in process; and scheduling is done in a stochastic or deterministic environment. Most research has traditionally been focused on deterministic machine scheduling. This is because most machine scheduling decisions are at operational level and therefore it is reasonably assumed to be deterministic. Further, faster and reliable information flow due to high technology and shorter lead times provide strong support for deterministic scheduling in repetitive manufacturing environments.

1.2.2 Single Machine Scheduling

Single machine models are important for various reasons. The single machine environment is simple and a special case of all other environments. Single machine models often display properties that do not hold for either machines in parallel or machines in series. However, the results that can be obtained for single machine models provide a basis for heuristics for more complicated machine environments. In practice, scheduling problems in more complicated machine environments are often decomposed into subproblems that deal with single machines. For example, a complicated machine environment with a single bottleneck may give rise to a single machine model.

1.2.3 Machine Scheduling Objectives

The ultimate aim of any scheduler is to develop a feasible schedule that is optimal with respect to some objective. The first step in solving a scheduling problem is thus to define the scheduling objective. Ideally, the objective function should consist of all costs in the system that depend on scheduling decisions. In practice, however, such costs are often difficult to measure, or even to identify completely. Nevertheless, three types of decision-making goals seem to be prevalent in scheduling: efficient utilization of resources, average length of time spent by a job on the shop floor, and conformance of task completion times to prescribed deadlines.

The most common objective in machine scheduling is minimizing the *makespan*, i.e., the length of the schedule, which is concerned with efficient machine utilization. However, in some cases, the length of time an individual job stays in the shop may be more important because it has a direct impact on the inventory holding cost. Well studied objective functions in this case are sum of flow time and weighted sum of flow time. *Flow time* of a job is the time between the delivery of the finished job and the time when it becomes available for processing at the shop. Generally, different jobs have different holding costs and therefore, weights are assigned to jobs. Further, flow time is proportional to the inventory holding cost. Thus the *weighted flow time* of a job, which is the flow time multiplied by the corresponding weight, is equivalent to the inventory holding cost. In some environments, especially when delivery costs are significant, several jobs may be delivered together in a batch. When a cost is incurred on each batch delivery, it is better to increase the batch sizes so that the delivery cost is minimized. However, the larger batch sizes will lead for longer stays in the plant and cause larger inventory costs. Therefore, a tradeoff must be made between minimizing the total inventory cost and the delivery cost.

1.2.4 Machine Scheduling Algorithms and Complexity

Many scheduling problems have been viewed as optimization problems subject to constraints or as a combinatorial optimization problem. If a problem can be optimally solved using an efficient algorithm or mathematical models, then that problem is called an *easy* problem. However, many scheduling problems are *hard* problems for which finding an optimal solution efficiently is very difficult. These easy and hard problems are widely studied in a branch of computer science known as complexity theory.

Algorithms

An *algorithm* is a precise rule (or set of rules) specifying how to solve some problem. Usually the efficiency or complexity of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space or memory complexity) required to execute the algorithm.

An *Approximation* algorithm is an algorithm to solve an optimization problem that runs in polynomial time in the length of the input and generates a solution that is guaranteed to be close to the optimal solution. An $\alpha - approximation$ algorithm is a polynomial time algorithm which always produces a solution of value within α times the value of an optimal (minimum) solution. Thus, an approximation algorithm provides solutions that will be generally an upper bound for a given problem.

In some cases, a lower bound for a given problem is obtained by relaxing some constraints. For example, in a scheduling problem, we may relax the problem so that jobs can be preempted. The solution of this relaxed problem will be a lower bound for the original problem.

A *Heuristic* is a rule of thumb or educated guess that reduces the search for solutions in domains that are difficult and poorly understood. The word heuristic comes from the Greek root *euriskw* meaning *to discover*. Heuristics may not always achieve the desired outcome, but can be extremely valuable to problem-solving processes. Good heuristics can dramatically reduce the time required to solve a problem by eliminating the need to consider unlikely possibilities or irrelevant states. Neighborhood search algorithms have become a popular heuristic technique for solving difficult combinatorial optimization problems. The standard neighborhood search algorithm is an iterative procedure that starts at a feasible solution for the optimization problem at hand and looks for a better solution in the neighborhood of the current solution. If a better solution is found, the current solution is replaced with it and another iteration is started, otherwise the algorithm terminates. The main shortcoming of these classical heuristic methods is their inability to escape local optimality.

A number of *meta-heuristic* approaches have been proposed to modify these classical neighborhood search algorithms to avoid convergence of the solution at local optimal points. Well known examples of meta-heuristics include simulated annealing, tabu search, genetic algorithm, and ant colony algorithm. We use a genetic algorithm to solve one of the manufacturer's problems discussed in Chapter 5.

Genetic Algorithm (GA) is inspired by the efficiency of natural selection in biological evaluation. GAs have been applied successfully to a wide variety of combinatorial optimization problems. Unlike classical heuristics such as tabu search and simulated annealing that generate a single solution and work hard to improve it, GAs maintain a large number of solutions and perform comparatively little work on each one.

As discussed earlier, a heuristic algorithm does not guarantee the optimality. Generally two standard methods are used to compare the performance of heurisctic algorithms: (i) develop an algorithm to obtain a lower bound for the problem, and compare the performance of the heuristic with the lower bound for some randomly generated problem instances; or (ii) develop an α-approximation algorithm and compare the performance of the heuristic with the approximation algorithm. In Chapter 5, we use the former method to compare the performance of the algorithm we developed.

Complexity

The term complexity refers to the computing effort (measured by the number of elementary computations required as a function of the order-of-magnitude of the input data) required by an algorithm. An algorithm is said to be a polynomial time algorithm if its number of computations is polynomial in input data size. According to complexity theory, any optimization problem can be categorized as an easy problem or a hard problem. Proving a problem is hard itself is very difficult. Thus, computer scientists use more powerful methods for proving a problem is hard. They first analyze the decision version of the given problem. A *decision problem* is one whose solution is either *yes* or *no*. For example consider the optimization problem of traveling salesman problem where we want to find the tour of visiting all the cities once with minimum travel length. The decision problem is then, given a set of cities, the distances between cities, and a bound k, does there exist a tour of all the cities having total length k or less? If someone gives a 'yes' answer to the decision problem with the corresponding tour, we may verify it by adding all the distances of the corresponding tour. If such verification for a 'yes' instance to the decision problem can be done in polynomial time then that problem belongs to the class NP (non-deterministic polynomial time).

Run time of an algorithm depends on the encoding of the input/output. There are two main encoding systems in use. In *unary* numeral system, a number is represented by a string of multiple instances of an arbitrarily chosen symbol. For example, if we choose the symbol *, then the number 6 is represented as ****** and therefore, the size of the input is $O(6)$. In *binary* numeral system, a number is represented by a radix of two, i.e., each digit will have one of two different values. Typically the symbols 0 and 1 are used to represent binary numbers. For example, the number 6 is represented as 110 and the size of the input is therefore, $O(\lceil log6 \rceil)$.

Decision problems that can be solved (on a deterministic sequential machine) in an amount of time that is polynomial in the size of the input are called easy problems which belong to the class P(polynomial time). Decision problems for which answers can be verified in polynomial time and no other NP problem is harder are called NP-Complete (NPC). When the decision version of an optimization problem belongs to the NP-Complete class, then the optimization problem is called NP-hard. NPC consists of the hardest problems in NP. The reason is that if one could find a way to solve an NP-hard problem quickly, then we could use that algorithm to solve all NP problems quickly. At present, all known algorithms for NP-hard problems require time that is exponential in the input size. Therefore, for NP-hard problems, the search for an efficient, exact algorithm should be given low priority and other less ambitious approaches must be given more priority. Thus, the knowledge that a problem is NP-hard provides valuable information about what approach has the potential of being most productive. Approximation algorithms and heuristics are mainly used approaches to solve NPC problems.

Decision problems which are still NP-hard even when all numbers in the input are bounded by some polynomial in the length of the input are called *strongly* NP-hard. NP-hard problems which are not NP-hard when all the numbers are bounded by some polynomial in the length of the input are called *weakly* NP-hard.

The existence of many scheduling models and their complexities has made the scheduling field a focal point for the development, application, and evaluation of combinatorial procedures and heuristic solution approaches. The selection of an appropriate technique depends mainly on the nature of the model and the choice of objective function. Many scheduling problems are proved to belong to NPC. Since it is hard to develop an efficient algorithm for NP-hard problems, generally approximation algorithms, heuristics, and simulation techniques are used to solve such scheduling problems.

1.2.5 Batch Scheduling

In traditional manufacturing systems, efficient utilization of resources such as machines, transportation vehicles and labor were considered the key issues for their successful operation. For example, when there is a setup required for each product on a machine, it is assumed that products must be processed in single lots so that the machine is utilized efficiently. Another example is, a whole lot is produced and sent to downstream customers, in order to fully utilize the transportation vehicle on a single trip. Therefore in traditional scheduling problems, selecting lot sizes and sequencing products were the major issues.

Larger lot sizes may be attractive due to fewer setups, less loss of production time, higher utilization of resources, more throughput and less time required to process all the operations. On the other hand, a smaller batch may prevent an important operation from waiting for a prolonged time for a different setup. Smaller batches may also reduce storage space requirements, the amount of capital tied up in inventory, and the average lead time. Thus researchers realized that the lot sizes must be further split into small batches so that machine idle time can be minimized, lead time can be reduced, and inventory holding cost can be minimized, while considering the high setup cost and/or the delivery cost. Further, recent trends in manufacturing, such as increasing demand toward customized products; increasing competition for market share from both domestic and international manufacturers; changing manufacturing technology; and changing customer needs and shorter product life cycles, force manufacturing organizations to focus on small batch production [26].

Modern technologies of flexible manufacturing reduce the setup cost/time and thus, provide an opportunity to process jobs in small batches. Thus *Batch scheduling* problems, as combinations of sequencing and partitioning problems, have been of great interest over the past two decades. Generally, the batching component converts sales orders into the form required for a specific production environment. As in many cases, there are different versions of batching problems that exist according to the nature of the batch. In *discrete* version, batches can have only integer number of items, whereas in *continuous* version, a batch can have a fraction of items. There are different versions of batching problems depending on the calculation of the length of the batch processing time. For *s-batching* (serial batching) problems, the length is the sum of the processing times of the jobs in the batch, whereas for *p-batching* (parallel batching) problems, the length is the maximum of the processing time of the jobs in the batch. This thesis studies s-batching problems.

1.3 Supply Chains

Fierce competition in today's global markets, the introduction of products with short life cycles, and the heightened expectations of customers have forced business enterprises to invest in, and focus attention on, their supply chains. In a typical supply chain, raw materials are procured, items are produced at one or more factories, shipped to warehouses for intermediate storage, and then shipped to retailers or customers. Therefore, a supply chain consists of suppliers, manufacturing centers, warehouses, distribution centers, and retail outlets. Besides these there are raw materials, work-in-process inventory, and finished products that flow between the facilities in a supply chain.

The objective of a supply chain is to be efficient and cost-effective across the entire system. Thus the total system-wide costs, from transportation and distribution to inventories of raw materials, work in process, and finished goods, are to be minimized. Through supply chain integration a firm can significantly reduce costs and improve service levels. Unfortunately, supply chain integration is difficult for two main reasons: different entities in the supply chain may have conflicting objectives and a supply chain is a dynamic system that evolves over time.

1.3.1 Push and Pull Systems in Supply Chains

In a *push* system, products are being produced without being preassigned to any particular customer, i.e., products are produced based on forecasted demand and pushed through the system. In a *pull* system processes are based on customer demand, i.e., each process is manufacturing each component in line with the next downstream department's needs to build a final part to the exact expectation of delivery from the customer. Thus the demand for the final product generates implied upstream demands for parts and components in earlier stages of the production process, i.e., demand pulling the whole system. Push systems work well in environments where there are high customer demands and quick product turnaround times. It is often argued that in a push system too much WIP results in waste and failure to meet production targets. However, pull systems may not be suitable for all business types because of product types, lead times and any stock holding arrangements with customers. In an efficient supply chain, the whole system may be neither a pure push system nor a pure pull system but a hybrid system. Generally the upstream suppliers or entities follow a push system and the downstream suppliers or entities follow a pull system. The entity at which market pull meets push is called the *decoupling point.* Thus in the supply chain, the real demand penetrates upstream up to the decoupling point. Beyond that the forecasted demand is used for scheduling. Items that are kept in stock at the upstream of the decoupling point are those items for which demand must be forecasted due to the fact that future demand between the moment of release of items and the moment those items are received is (partially) unknown, i.e., the lead time of supplying the item is longer than the lead time requested by the customer. At the downstream of the decoupling point, items are normally not kept in stock since future demand for these items between release moments and receipt moments is known, i.e., the lead time of supplying the item from the decoupling point to the customer is shorter than the lead time requested by the customer. We study batch scheduling at an entity when it is in the push system and when it is in the pull system in the supply chain.

1.4 Demand Driven Supply Chains

In this highly competitive and demanding era, suppliers are forced to meet the demands at the right time. Thus, demand driven supply chains are mainly controlled by consumer needs and/or wants. This in turn calls for Just-In-Time (JIT) manufacturing at each supplier. *Just-In-Time* manufacturing is a management philosophy that strives to increase value added and eliminate sources of manufacturing waste by producing the right parts in the right quantities at the right time [26].

A classical push production system may not improve supply chain performance even if perfect information flow among partners is achieved. On the other hand, JIT may be just the right mode of organizing the production process to take full advantage of an essential part to benefit from supply chain coordination and perfect information flow. One of the key points of successful JIT operations is maintaining low inventory levels while meeting the customer demands on time. To achieve this, it needs a synchronized movement of inputs and outputs in the production and delivery of goods and services to customers. Thus the competitive success of an organization which follows the JIT principle no longer depends only on its own efforts, but relies on the efficiency of the entire supply chain.

1.5 Supply Chain Scheduling

Supply chain management has been one of the most important topics for researchers over the last fifteen years. In traditional scheduling, researchers are mainly concerned with economy in production and distribution, and efficiency in resource utilization. Later they realized that effectiveness must be given the most priority in scheduling. The effectiveness of a schedule depends on the schedules of other entities in the supply chain. Thus a proper coordination would give more effective schedules. Supply chain scheduling, a recently emerging research area in scheduling, is concerned with the coordination among the entities of a supply chain.

As it is a new area of research, there are relatively few works dealing specifically with scheduling problems in supply chains. Depending on the number of stages in the supply chain, operating it may involve decisions by several decision-makers, e.g., the supplier and the customers. This gives rise to a rich variety of optimization problems for each decision-maker, and problems of coordination between them.

1.6 Problem Definition

We study batch scheduling problems at the supplier, manufacturer, and supplier-manufacturer pair considered as a single system in a push environment, and batch scheduling problems in a pull manufacturing system which we call *demand driven supply chain*. In our analysis, we represent an entity (such as a supplier or a manufacturer) in the supply chain by a single machine. Figure 1.1 shows a supply chain model where a single supplier S delivers products to m manufacturers $M_1, M_2, \ldots, M_m$ and the manufacturer M_1 delivers products to h end-customers $C_1, C_2, \ldots, C_h$.

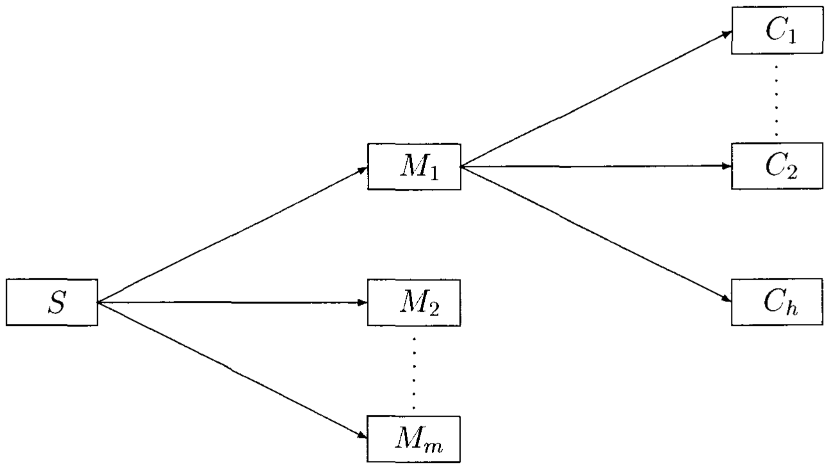


Figure 1.1: A supply chain with 1 supplier, m manufacturers, and h customers.

We assume that batch delivery cost does not depend on the batch size but on the customer to which it is destined for, and the batch processing time is the sum of processing times of items that compose the batch. We further assume that a holding cost is incurred for each job from its availability to its delivery, and the machine is continually available and can process at most one item at a time.

We focus on finding the optimal production batch sizes that would minimize the inventory holding cost and the batch delivery cost. As we discussed earlier, flow times have direct impact on inventory holding costs. Thus, we study batch sizes which will minimize the weighted sum of flow times and the total delivery cost.

Batch Scheduling at the Supplier

Batch scheduling problems at the supplier are discussed in Chapter 3 and Chapter 4. We assume that all the items are available at time zero, i.e., at the start of the planning horizon. This is a reasonable assumption at an upstream supplier where holding cost of raw materials and/or components is not expensive. Chapter 3 studies polynomial algorihms for some batch scheduling problems, whereas Chapter 4 studies an NP-hard problem and develops a 2-approximation algorithm for it.

Batch Scheduling at the Manufacturer

Chapter 5 analyses batch scheduling at the manufacturer. Unlike in the supplier's problem, jobs arrive in batches at the manufacturer. Therefore, jobs have release time. We provide polynomial algorithms, approximation algorithms, and heuristic algorithms for different types of batch scheduling problems at the manufacturer.

Batch Scheduling in a Demand Driven Supply Chain

We assume that the part manufacturer gets components from its only supplier for each product, processes them and sends the processed parts to its immediate customers. Customers order jobs at the manufacturer with required quantity and delivery time specified and the manufacturer has to deliver the right quantities at the promised delivery times. Since it is a demand driven supply chain, the manufacturer in turn specifies job requirements and arrival dates to its supplier and the supplier will deliver the right quantities at the promised delivery times to the manufacturer. We further assume that since the customer of a transaction decides the delivery time and the batch sizes of its immediate supplier, the delivery cost is charged partly or in whole to the customer. Therefore, the manufacturer's problem is to find the optimal arrival batch sizes and the corresponding arrival times from the supplier so that the total flow time related cost and the batch delivery cost is minimized while delivering the jobs to customers at the promised delivery times. Chapter 6 studies this problem. We develop dynamic programming algorithms to solve batch scheduling problems in demand driven supply chains. We will also consider the total cost minimization of weighted sum of flow time or some due date constraints with delivery cost.

Supplier Manufacturer Coordination and Batch Scheduling

Chapter 7 studies supplier manufacturer coordination and batch scheduling. We consider the supplier and the manufacturer as a single system and study the batch scheduling problems so that the total cost of the system is minimized. It is clear that this problem is harder than the batch scheduling problem at the supplier and manufacturer. We develop a basic model where a single supplier delivers products to a single manufacturer and the manufacturer processes and delivers products to a single customer. We develop two dynamic programming algorithms to solve the batch scheduling in the system under two conditions.

1.7 Some Applications

The problems studied in this thesis have many practical applications and we provide a few examples.

Consider a manufacturer producing plastic containers for different customers who make juice, milk, vegetable oil, shampoo etc. Each customer has its own shape, size, colour and texture for the containers. When switching from one product to another one a setup is required. The delivery costs to different customers vary based on their locations. The manufacturer, therefore, has to find the optimal job sequence, number of batches, and batch sizes so that the total cost of inventory holding and delivery is minimized. We present a polynomial-time solution for this problem.

Generally, the pasta industry involves manufacturing long goods and short goods. Spaghetti, capilli, linguini, angel hair etc are long goods, and macaroni, rigatoni, fusilli etc are short goods. There are also many novelty shapes such as bow ties, shells, cannolloni, lasagna and wheels. In addition, some pastas now include spinach and other vegetables. Pasta manufacturers produce these different products and deliver to their customers. The manufacturer has to decide batch sizes so that the total inventory holding cost and delivery cost is minimized.

The study on batch scheduling in customer centric supply chains is motivated by the wide adoption of JIT systems in many successful production organizations. One example is BMW, the winner of the productivity award for manufacturing by Modern Materials Handling [6]. BMW's newly expanded manufacturing plant in Spartanburg, S.C., uses a pull system to build customer-specified vehicles within 10 days of order placement. Although the plant builds only two models, X-5 sports utility vehicles and the two-seater Z-4 roadster, there are many options available for each model in terms of shape, colour, and interior requirements. For example, for the X-5 model, there are 8 body variances, 12 colours, 19 engine choices, 16 interior choices, and 85 other options. The plant keeps its suppliers constantly informed of accurate and stable demand data. As a result, the plant is able to follow the JIT philosophy successfully.

1.8 Motivation

A supply chain consists of suppliers, manufacturers, and customers, i.e., many manufacturing enterprises. Each enterprise may have multi-stage processing in a serial, parallel, or assembly model. Thus, supply chain coordination and scheduling is a very complex problem to solve. Therefore, as Bhatnagar et. al [5] discussed, we need to represent each enterprise of the supply chain by a simpler system which captures the salient features of the original enterprise, and then develop a suitable lotsizing technique which can be applied to the simplified system. In our work we consider each entity as a single machine. The single machine representation is reasonable in many cases because it may be possible to solve the embedded single-machine problem independently and then incorporate the result into the larger problem. For example, there may be a bottleneck stage in the multi-processes of an enterprise, and therefore treating the bottleneck as a single machine and analysing the single machine problem may provide some insights about the enterprise.

Further, some decisions treat resources in the aggregate, as if jobs were coming to a single facility. In order to completely understand the behavior of a complex system, it is vital to understand its parts. Thus analysis of batch scheduling at the supplier and manufacturer will give some ideas to solve the coordination and batch scheduling at a supplier-manufacturer pair.

As discussed earlier, a trade-off between inventory holding cost and the delivery cost is very important in supply chain scheduling. Thus we study batching and scheduling of jobs on a single machine to minimize the sum of weighted flow time, and the total delivery cost, where weight assigned to each job is generally assumed to be the unit holding cost of that job. When all the jobs have the same unit holding cost, minimizing the sum of flow times will be equivalent to minimizing the weighted sum flow times.

Chapter 2

Literature Review

2.1 Single Machine Batch Scheduling

In the mid 1980s researchers found that the inventory holding cost of WIP inventory in a shop contributed a considerable portion of an organization's total cost. Such a large cost proportion has attracted many researchers to find a compromise between full utilization of resources and WIP inventory reduction. Batching is a well-known model for this situation. *Batching* is to split a lot into batches and to schedule these batches for processing. The *lot size* is a predetermined quantity typically set by the customer or by the planning processes that precede scheduling. A *batch* is a set of items (jobs) which must be processed jointly. For a given batch the number of items it contains is called its *batch size.*

Potts and Van Wassenhove [32], Albers and Brucker [1], Webster and Baker [41], and Potts and Kovalyov [31] gave comprehensive surveys on batch scheduling. Several papers have dealt with one-machine batch scheduling problems in which the objective is to minimize the inventory holding cost. Dobson et al. [16], Santos and Magazine [34], Naddef and Santos [28] and Coffman et al. [14] have studied the batching of identical items processed on a single machine to minimize the inventory holding cost when each batch requires a setup. They give explicit formulas for determining the optimal number of batches to be used. Finally, Shallcross [35] gave a polynomial-time solution for the problem. Dobson et al. [16], and Naddef and Santos [28] also propose heuristic solutions for the corresponding N-job problem under the assumption that only items of the same job (product) can be part of a batch. Monma and Potts [27] study the batching problem in a one-machine environment for N products where a batch may contain items from different jobs. They prove that jobs within each batch are sequenced in shortest-weighted-processing-time order in the optimal batching to minimize the inventory holding cost. There are only a few studies which consider the delivery cost in batching problems. Among these, Cheng et al. [12] study the batch scheduling problem on a single machine to minimize the sum of delivery costs and earliness penalties. Yang [43] analyzes a similar model with given batch delivery dates. Chen [9] presents a dynamic programming algorithm for single machine scheduling and common due date assignment with earliness and/or tardiness penalties and batch delivery costs.

Most papers on batching problems consider the single product problems or in the multi-product case, the optimal batch sizes are obtained for a product sequence which is assumed to be given. Although this hierarchical solution scheme of the multi-product case is appealing, it is not necessarily optimal.

2.2 Scheduling Problems and Genetic Algorithms

Genetic Algorithm (GA) is a meta-heuristic inspired by the efficiency of natural selection in biological evoluation. GA was introduced by Holland [22] as a method for modeling complex systems. The general idea of GAs is to start with randomly generated solutions and, implementing a "survival-of-the-fittest" strategy, evolve good solutions. Even though, GA was introduced more than three decades ago, it has seen impressive growth in the past decade. GA is an iterative procedure that consists of a constant size population of individuals, each one represented by a finite string of symbols, known as genome, encoding a possible sollution in a given problem space. The standard GA proceeds as follows: (i) an initial population of individuals is generated at random or heuristically. (ii) every individual in the population is decoded and evaluated according to some predefined quality criterion, referred to as fitness. (iii) individuals are selected, to form a new population, according to their fitness. (iv) genetically inspired operators are used to introduce new inividuals into the population. (v) terminal condition is specified as some fixed, maximal number of generations or as the attainment of an acceptable fitness level for the best individual.

There are many selection procedures currently in use, one of the simplest being Holland's original fitness-proportionate selection, where individuals are selected with a probability proportional to their relative fitness. This ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population.

The best known operators are crossover and mutation. Crossover is performed, with a given probability p_{cr} between two selected individuals, called parents, by exchanging parts of their genomes to form two new individuals, called offsprings. In its simplest form substrings are exchanged after a randomly selected crossover point. This operator tends to enable the evolutionary process to move towards "promising" regions of the search space. If the crossover operations is not performed, with probability $(1 - p_{cr})$, then the offsprings are exact copies of each parent. The mutation operator is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space. Mutation entails flipping bits at random, with some small probability p_m.

Since the publication of a paper by Davis [15], a lot of research has been done in the field of production scheduling with GAs. The main difficulty in this subject arises from the question of how to represent the problem in the algorithm, which is the most important for genetic search. A tutorial survey of the representation approaches in the literature can be found in Cheng et. al. [11].

We present a *Random Keys Genetic Algorithm* (RKGA) to solve one of our problems in Chapter 5. Our approach is based on the random keys encoding of Bean [4]. This method encodes a solution with a random number. These values are used as sort keys to decode the solution. The RKGA operates in two spaces, the chromosome space and the schedule assignment space. We briefly discuss the RKGA procedure in Chapter 5 when developing the meta heuristic algorithm.

2.3 Coordination and Scheduling in Supply Chains

Supply chain management is a major issue in many industries as firms realize the importance of creating an integrated relationship with their suppliers and customers. The new trends in IT, the ever changing taste of customers, as well as quality and price pressure compel firms to focus on their supply chains, i.e., to work in an integration of organizations that cooperate for improving the flow of material and information from supplier to customers at the lowest cost and highest speed. Thomas and Griffin [39] provide an extensive review and discussion of the supply chain management literature. They point out that for many products logistics expenditures can constitute as much as 30% of their cost. Whereas most of the supply chain literature focuses on inventory control issues on the strategic level, using stochastic models, Thomas and Griffin [39] discuss the need for research dealing with supply chain problems on the *operational* level instead, using *deterministic* rather than stochastic models.

Supply chain coordination is driven by the realization that the typical marketing channel contains redundant activities and unnecessary inventories. To eliminate these redundant activities and unnecessary inventories, manufacturers must take a channel-wide perspective concerning inventories. The purpose of supply chain coordination is to integrate the output of a firm's supply chain efforts with other components of manufacturing and the marketing mix so that customer satisfaction can be maximized and competitive advantage can be achieved across all levels in the supply chain. In recent years there has been an increasing focus on the integration of different segments of the supply chain. Chandra and Fisher [8] study integration and coordination of production and distribution functions. Bhatnagar et al. [5] study two broad levels of coordination, general coordination and multi-plant coordination. General coordination is the integration of different functions, whereas the multi-plant coordination is the integration among the plants of an internal supply chain. The authors claim that an efficient coordination will be possible only when the effects of uncertainty of final demand, uncertainties of production processes at each plant, and capacity constraints at each plant are taken into consideration. Treville et. al [40] claim that the efforts for supply chain coordination between partners without the effort for lead time reduction may not improve the chain's performance to its best.

Hall and Potts [19] study batch scheduling in supply chains. They study the coordination of a supplier-manufacturer pair and conclude that the coordinated batching decision will really provide the lowest system cost. They study a variety of scheduling, batching and delivery problems in supply chains with the objective of minimizing the overall scheduling and delivery cost. Chen and Hall [10] extend these to supply chains with assembly-type manufacturing systems. Some of the issues studied in these papers are related to previous work on coordinating production and distribution systems. We mention here the papers by Williams [42], and Lee and

Chen [24], which consider the integration of transportation time and capacity issues with scheduling decisions.

2.4 Demand Driven Supply Chain

Manufacturers are caught between increasing customer demands, more intense global competition, shorter product lifecycles, and growing supply chain complexity. Therefore, many manufacturers are turning to demand-driven manufacturing for their survival in the market. The basic concept of demand-driven manufacturing is that customer demand signals drive manufacturing plans and operations. Customers pull product from suppliers as needed, instead of suppliers pushing product to customers in anticipation of forecasted demand. Thus, in demand driven supply chain, providing the right quantity of the right product at the right time is very important.

There have been many works in single machine scheduling with due dates as constraints. Some of the major objectives, which are related to due dates are sum of tardinesss, maximum lateness, sum of weighted tardiness, sum of weighted earliness/tardiness, and sum of tardy jobs. These due date related objectives consider mainly minimizing deviation from due dates. Also several other papers have studied single machine scheduling problems with deadline constraints where jobs must be delivered on or before the due date. Smith [37] develops a polynomial time algorithm for the sum of the completion time problem with deadline constraints and Heck and Roberts [20] present an algorithm for minimizing the sum of completion times subject to not increasing the maximum tardiness. Emmons [17] and Burns [7] provide counter examples to show that the extension of Smith [37] and Heck and Roberts [20] for arbitrary positive weights will not work. Burns [7] also develops a new algorithm to get local optima for the arbitrary weight case. Bansal [3] proposes a branch and bound algorithm with dominance criteria for the total weighted completion time problem with deadline constraints. Potts and Van Wassenhove [33] improve Bansal's procedure using additional dominance criteria. Posner [30] provides a branch and bound algorithm with a lower bound calculated for a preemptive version. Then Bagchi and Ahmadi [2] provide a lower bound which is better than Posner's lower bound. Finally Pan [29] provides an efficient branch and bound algorithm to solve weighted completion time with deadline constraints. Steiner and Stephenson [38] study the closely related problem of minimizing weighted completion time and maximum lateness. All these papers accept early deliveries, assume job arrivals to the shop are not decision variables, and assume negligible delivery cost of jobs to customers.

Chapter 3

Polynomial Algorithms for the Supplier's Problem

3.1 Introduction

In this chapter, we analyse different problems of the supplier which can be solved optimally in polynomial time. Batch scheduling problems are combinations of sequencing and partitioning problems. There have been many works which develop batch scheduling algorithms using a hierarchical procedure where they first assume a job sequence and then find the batching of this job sequence. This hierarchical procedure may not guarantee optimality. However, if we could prove that job sequencing and batching are separable, then we could use the hierarchical procedure to solve our batching problem optimally, if there is a polynomial algorithm to find the job sequence and a polynomial algorithm to batch a given job sequence. For all the problems we study in this chapter, we use the above property to obtain the optimal batching schedule. We prove that problems with single product, with jobs of identical weights, and with jobs of identical processing times can be solved polynomially and develop algorithms for these problems.

We study the batch scheduling problem at the supplier where production is controlled by a push system. There are m manufacturers $M^{(1)}, M^{(2)}, \cdots, M^{(m)}$ with $N^{(j)}$ number of jobs for customer $M^{(j)}$ $(j = 1, 2, \cdots, m)$. We refer to these manufacturers simply as customers in this Chapter and in Chapter 4. The supplier is required to process jobs on a single machine and the kth job of customer $M^{(j)}$ denoted by $J_k^{(j)}$ needs $p_k^{(j)}$ time to process on the machine. All the jobs are available at time zero, i.e., at the start of the planning horizon. This is a reasonable assumption at the upstream suppliers when production is controlled by push system. We further assume that the batch delivery cost is partially or in whole is charged to the supplier.

Associated with each batch delivery to customer $M^{(j)}$ is the delivery cost $d^{(j)}$. A holding cost of $w_k^{(j)}$ incurs per unit time for job $J_k^{(j)}$. We also use the term *weight* for $w_k^{(j)}$. We have to find the batch schedule so that the total cost of inventory holding and batch delivery at the supplier is minimized, i.e., our objective is to minimize $TC = \sum_{j=1}^{m} \sum_{i=1}^{n^{(j)}} W_i^{(j)} C_i^{(j)} + \sum_{j=1}^{m} n^{(j)} d^{(j)}$, where $n^{(j)}$ is the number of batches for customer $M^{(j)}$, $B_i^{(j)}$ is the ith batch to customer $M^{(j)}$, $C_i^{(j)}$ is the completion time of batch $B_i^{(j)}$, and $W_i^{(j)}$ is the sum of the weight of all the jobs assigned to batch $B_i^{(j)}$.

For convenience, we remove the subscript k (resp., superscript (j)) when we deal with single-product (resp., single-customer) problems. Chapter 3 analyses some polynomially solvable problems and Chapter 4 develops a 2-approximation algorithm for the multiple product batch scheduling problem of the supplier which is known to be NP-hard.

First, we make some observations that are used later in our proofs. We omit the proofs of these since they are very simple.

Observation 1: There should not be any idle time on the machine during the whole production horizon.

Observation 2: A batch must be delivered as soon as the processing of all the items belonging to that batch is completed.

Observation 3: Items belonging to the same batch must be processed consecutively.

3.2 Single Product Single Customer Problem

We study the basic batching problem at the supplier, i.e., the problem with single product and single customer. In the single product problem, $w_i = w, p_i = p$ for $i = 1, 2, \ldots, N$. We first prove some properties for the $n-$*batching problem* in which the number of batches is fixed at n, and for the general optimal batching problem with variable number of batches. Then we show that this basic model can be solved in polynomial time. We denote the ith batch and its size by b_i.

The following observation is used for this problem.

Definition 1: If N items are split into n batches of integer size b_i such that $b_i \in \{x, x-1\}$ for $i = 1, 2, ..., n$, then $x = \lceil \frac{N}{n} \rceil$. We refer to this as the *almost-equal-batch-size (AEBS)* policy.

3.2.1 The n-batching Problem

In the n-batching problem, N identical items must be split into n batches, for a given n, so that $b_i > 0$ for $i = 1, 2, ..., n$, and the total holding cost of all the batches is minimized. In this section, we derive expressions for the optimal batch sizes for the $n-$batching problem. This will lead to a polynomial-time solution of the overall problem.

Property 3.1 *Given batch sizes $b_1, b_2, ..., b_n$, the order of the batches does not affect*

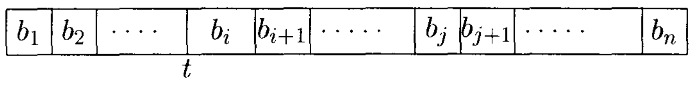


Figure 3.1: n-batching schedule

the objective value.

Proof: Let the holding cost of batch sequence $b_1, b_2, ..., b_n$ be H_n. Then

$$\begin{aligned} H_n &= w(b_1(b_1)p + b_2(b_1+b_2)p + ... + b_n(b_1+b_2+...+b_n)p) \\ &= pw(b_1(b_1) + b_2(b_1+b_2) + ... + b_n(b_1+b_2+...+b_n)). \end{aligned}$$

This is a symmetric function of the batch sizes $b_1, b_2, ..., b_n$. Therefore, the order of the batches does not affect the holding cost. Delivery cost is fixed at nd. Hence, the total cost is not affected by the batch sequence. □

Lemma 3.1 shows that the AEBS policy is optimal in the single-product single-customer case.

Lemma 3.1 *The AEBS policy is optimal for the n-batching problem with* $b_i \in \{\lceil \frac{N}{n} \rceil, \lceil \frac{N}{n} \rceil - 1\}$, *for* $i = 1, 2, ..., n$.

Proof: Figure 3.1 shows a typical n-batching schedule. Let the ith batch start at time t, and consider batch b_j for some $j > i$. Let the holding cost of the batches $(b_1, b_2, ..., b_{i-1}, b_{j+1}, b_{j+2}, ..., b_n)$ be denoted by A and the holding cost of the whole schedule by I. Then I equals the holding cost of $(b_i + b_{i+1} + ... + b_j)$ plus A. Thus we can write

$$\begin{aligned} I &= b_i w(t + b_i p) + b_{i+1} w(t + (b_i + b_{i+1})p) + ... + b_j w(t + (b_i + b_{i+1} + ... + b_j)p) + A \\ &= tw(b_i + b_{i+1} + ... + b_j) + pw(b_i(b_i) + b_{i+1}(b_i + b_{i+1}) + ... + b_j(b_i + b_{i+1} + ... + b_j)) + A \end{aligned}$$

Now consider the schedule in which one item is moved from b_i to b_j. Let the holding cost of this schedule be I'. The holding cost of the batches $(b_1, b_2, ..., b_{i-1}, b_{j+1}, b_{j+2}, ..., b_n)$

clearly remains A. Therefore,

$$I' = tw(b_i + b_{i+1} + ... + b_j) + pw((b_i - 1)(b_i - 1) + b_{i+1}(b_i - 1 + b_{i+1}) + ...$$
$$+ b_{j-1}(b_i - 1 + b_{i+1} + ... + b_{j-1}) + (b_j + 1)(b_i - 1 + b_{i+1} + ... + b_{j-1} + b_j + 1)) + A$$

After some simple calculations, we obtain $I - I' = pw(b_i - b_j - 1)$.

Thus, there will be savings in the holding cost by moving one item from b_i to b_j, if $I - I' > 0$, i.e., if $b_i - b_j > 1$. Since this does not change the number of batches, the delivery cost part for the schedule will not change. By Property 3.1 , the sequence of the batches does not affect I. Thus, repeating the above argument for a sequence in which b_j and b_i are interchanged, we obtain that there will be savings by moving one item from b_j to b_i if $b_j - b_i > 1$. This implies that all the batch sizes will be either x or $(x-1)$ for some integer x in the optimal schedule. Thus, by Definition 1, $b_i \in \left\{ \lceil \frac{N}{n} \rceil, \lceil \frac{N}{n} \rceil - 1 \right\}$ for $i = 1, 2, ..., n$ in the optimal batching.

To be more precise, assume that we have l batches of size $\lceil \frac{N}{n} \rceil$ and $n-l$ batches of size $\lceil \frac{N}{n} \rceil - 1$. This means that that we must have $l \cdot \lceil \frac{N}{n} \rceil + (n-l) \cdot \left(\lceil \frac{N}{n} \rceil - 1 \right) = N$. Solving this for l, we obtain that there will be $l = N - n \cdot \left(\lceil \frac{N}{n} \rceil - 1 \right)$ batches of size $\lceil \frac{N}{n} \rceil$ and the remaining batches will have size $\lceil \frac{N}{n} \rceil - 1$ in the AEBS solution, which can be obtained in constant time for a given n. □

Lemma 3.2 *The holding cost of the optimal n-batching schedule is*

$$H_n^* = \frac{pw}{2}\left(2N \left\lceil \frac{N}{n} \right\rceil - n \left\lceil \frac{N}{n} \right\rceil^2 + n \left\lceil \frac{N}{n} \right\rceil + N^2 - N\right).$$

Proof: From Lemma 3.1, we know that in the optimal batching $b_i \in \{x, x-1\}$ for $i = 1, 2, ..., n$, where $x = \lceil \frac{N}{n} \rceil$. Let m batches contain x items and $(n-m)$ batches contain $(x-1)$ items in the optimal batching. Then $mx + (n-m)(x-1) = N \Rightarrow$ $m = N - nx + n$

By Property 3.1, we can assume that all m batches of size x are scheduled first and are followed by the $(n-m)$ batches of size $(x-1)$. Therefore,

$$\begin{aligned}
H_n^* &= pw(x \cdot x + x \cdot 2x + ... + x \cdot mx) + pw((x-1)(mx + (x-1)) \\
&+ (x-1)(mx + 2(x-1)) + ... + (x-1)(mx + (n-m)(x-1))) \\
&= pw\Big(\frac{m}{2}(m+1)x^2 + m(n-m)x(x-1) + \frac{(n-m)}{2}(n-m+1)(x-1)^2\Big) \\
&= pw\Big(\frac{(N-nx+n)}{2}(N-nx+n+1)x^2 + (N-nx+n)(nx-N)x(x-1) \\
&+ \frac{(nx-N)}{2}(nx-N+1)(x-1)^2\Big) \\
&= \frac{pw}{2}(2Nx - nx^2 + nx + N^2 - N) \\
&= \frac{pw}{2}\Big(2N\left\lceil\frac{N}{n}\right\rceil - n\left\lceil\frac{N}{n}\right\rceil^2 + n\left\lceil\frac{N}{n}\right\rceil + N^2 - N\Big). \qquad \square
\end{aligned}$$

3.2.2 Optimal Batching Problem

In this section, we discuss the problem of finding the optimal number of batches for the basic model. Following this, the batch sizes corresponding to the optimal number of batches can be obtained using Lemma 3.1.

We first prove that the total cost function is a discrete convex function. A function $f(x)$, defined on a non-empty interval S of the integers, is *discrete convex* if for every two integer points x_1 and x_2 in S and every α such that $0 \leq \alpha \leq 1$ and $\alpha x_1 + (1-\alpha)x_2$ is integer, we have $f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha)f(x_2)$. In order to prove the convexity of the total cost function, we consider its two components: the holding cost and the delivery cost. An example showing the cost function and its components is depicted in Figure 3.2.

Lemma 3.3 *H_n^* is a monotonically decreasing function of n.*

Proof: It is easy to see that when we move the last item of any batch i with $b_i > 1$ of the best n-batch schedule to a new batch, the holding cost will decrease. Therefore,

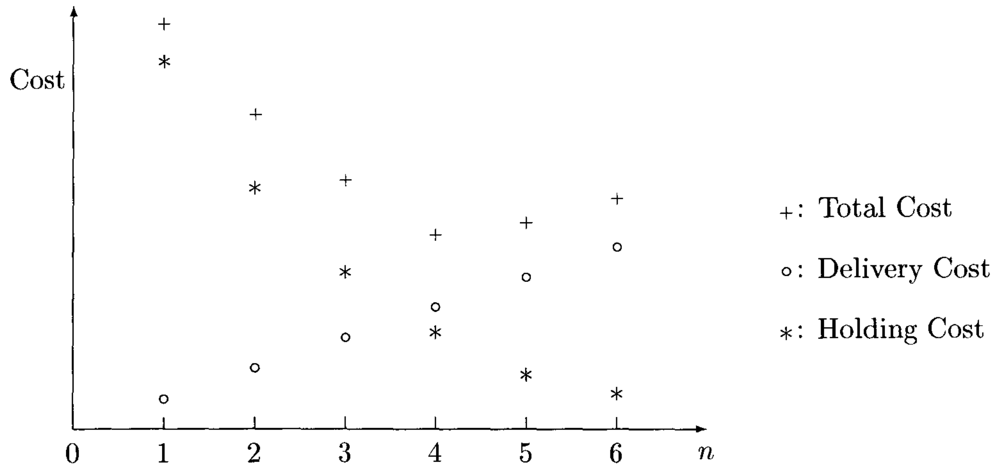


Figure 3.2: A Discrete Convex Cost Function

the holding cost of the best $(n+1)$-batch schedule will be smaller than that of the best n-batching. □

The *marginal savings* in the holding cost when we increase the number of batches from n to $n+1$ is defined by $H_n^* - H_{n+1}^*$. Lemma 3.4 proves the non-increasing behavior of the marginal savings of the holding cost.

Lemma 3.4 *The marginal savings of the holding cost function is non-increasing in n.*

Proof: The proof is based on extensive case analysis shown in detail below.

Let H_k^* be the holding cost of k-batching problems.
If we prove that $H_n^* - H_{n+1}^* \geq H_{n+1}^* - H_{n+2}^*$ for $1 \leq n \leq N-2$, then the marginal savings is non-increasing in n.

$$\begin{aligned}
H_n^* - H_{n+1}^* &= \frac{pw}{2}\left\{2N\left\lceil\frac{N}{n}\right\rceil - n\left\lceil\frac{N}{n}\right\rceil^2 + n\left\lceil\frac{N}{n}\right\rceil - 2N\left\lceil\frac{N}{n+1}\right\rceil + (n+1)\left\lceil\frac{N}{n+1}\right\rceil^2 - (n+1)\left\lceil\frac{N}{n+1}\right\rceil\right\} \\
H_{n+1}^* - H_{n+2}^* &= \frac{pw}{2}\left\{2N\left\lceil\frac{N}{n+1}\right\rceil - (n+1)\left\lceil\frac{N}{n+1}\right\rceil^2 + (n+1)\left\lceil\frac{N}{n+1}\right\rceil - 2N\left\lceil\frac{N}{n+2}\right\rceil\right. \\
&\quad \left. +(n+2)\left\lceil\frac{N}{n+2}\right\rceil^2 - (n+2)\left\lceil\frac{N}{n+2}\right\rceil\right\}
\end{aligned}$$

We have to show that $(H_n^* - H_{n+1}^*) - (H_{n+1}^* - H_{n+2}^*) \geq 0$. We show that $L - R \geq 0$ for all

n where,

$$L = 2N\left\lceil\frac{N}{n}\right\rceil + 2N\left\lceil\frac{N}{n+2}\right\rceil + n\left\lceil\frac{N}{n}\right\rceil + (n+2)\left\lceil\frac{N}{n+2}\right\rceil + 2(n+1)\left\lceil\frac{N}{n+1}\right\rceil^2 \text{ and}$$
$$R = 4N\left\lceil\frac{N}{n+1}\right\rceil + 2(n+1)\left\lceil\frac{N}{n+1}\right\rceil + n\left\lceil\frac{N}{n}\right\rceil^2 + (n+2)\left\lceil\frac{N}{n+2}\right\rceil^2$$

We consider the problem into different cases, and prove that $L - R \geq 0$ for all the cases.

We know that $N \geq n+2$. Let $N = a(n+1) + b$, where $a \geq 1, 0 \leq b \leq n$ and $a + b \geq 2$.

$$\left\lceil\frac{N}{n}\right\rceil = a + \left\lceil\frac{a+b}{n}\right\rceil; \left\lceil\frac{N}{n+1}\right\rceil = a + \left\lceil\frac{b}{n+1}\right\rceil; \text{ and } \left\lceil\frac{N}{n+2}\right\rceil = a - \left\lfloor\frac{a-b}{n+2}\right\rfloor$$

Case 1: $b = 0 \Rightarrow N = a(n+1); a \geq 2$

$\lceil\frac{N}{n}\rceil = a + \lceil\frac{a}{n}\rceil; \left\lceil\frac{N}{n+1}\right\rceil = a; \left\lceil\frac{N}{n+2}\right\rceil = a - \left\lfloor\frac{a}{n+2}\right\rfloor$

Case 1.1: $2 \leq a \leq n$

$\lceil\frac{N}{n}\rceil = a + 1; \left\lceil\frac{N}{n+1}\right\rceil = a; \left\lceil\frac{N}{n+2}\right\rceil = a.$

$$L = 2N(a+1) + 2Na + n(a+1) + (n+2)a + 2(n+1)a^2$$
$$R = 4Na + 2(n+1)a + n(a+1)^2 + (n+2)a^2$$
$$L - R = 2N - 2an = 2a(n+1) - 2an = 2a > 0$$

Case 1.2: $a = n+1$

$\lceil\frac{N}{n}\rceil = a + 2; \left\lceil\frac{N}{n+1}\right\rceil = a; \left\lceil\frac{N}{n+2}\right\rceil = a.$

$$L = 2N(a+2) + 2Na + n(a+2) + (n+2)a + 2(n+1)a^2$$
$$R = 4Na + 2(n+1)a + n(a+2)^2 + (n+2)a^2$$
$$L - R = 4a(n+1) - 2n - 4an = 4a - 2n = 4(n+1) - 2n > 0$$

Case 1.3: $a \geq n+2$

Let $a = dn + e$, where $d \geq 1, 0 \leq e < n; d + e \geq 2$.

Let $a = f(n+2) + g$, where $f \geq 1, 0 \leq g < n+2$.

$\lceil\frac{N}{n}\rceil = a + d + \lceil\frac{e}{n}\rceil; \left\lceil\frac{N}{n+1}\right\rceil = a; \left\lceil\frac{N}{n+2}\right\rceil = a - f.$

Case 1.3.1: $e = 0 \Rightarrow d \geq 2$

$\left\lceil \frac{N}{n} \right\rceil = a + d; \left\lceil \frac{N}{n+1} \right\rceil = a; \left\lceil \frac{N}{n+2} \right\rceil = a - f.$

$$\begin{aligned} L &= 2N(a+d) + 2N(a-f) + n(a+d) + (n+2)(a-f) + 2(n+1)a^2 \\ R &= 4Na + 2(n+1)a + n(a+d)^2 + (n+2)(a-f)^2 \\ L - R &= 2Nd - 2Nf + nd - nf - 2f - 2nad - nd^2 + 2anf + 4af - nf^2 - 2f^2 \\ &= 2da(n+1) - 2fa(n+1) + a - (a-g) - 2nad - da + 2anf + 4af - f(a-g) \\ &= 2ad + 2af + a - (a-g) - da - f(a-g) = ad + fa + g + fg > 0 \end{aligned}$$

Case 1.3.2: $e > 0$

$\left\lceil \frac{N}{n} \right\rceil = a + d + 1; \left\lceil \frac{N}{n+1} \right\rceil = a; \left\lceil \frac{N}{n+2} \right\rceil = a - f.$

$$\begin{aligned} L &= 2N(a+d+1) + 2N(a-f) + n(a+d+1) + (n+2)(a-f) + 2(n+1)a^2 \\ R &= 4Na + 2(n+1)a + n(a+d+1)^2 + (n+2)(a-f)^2 \\ L - R &= 2Nd - 2Nf + 2N - nd - nf - 2f - 2nad - nd^2 + 2anf - nf^2 + 4af - 2f^2 - 2na \\ &= 2da(n+1) - 2fa(n+1) + 2a(n+1) - nd - nf - 2f - 2nad - nd^2 \\ &+ 2anf - nf^2 + 4af - 2f^2 - 2na \\ &= 2ad + 2af - nd - nf - 2f - nd^2 - nf^2 - 2f^2 + 2a \\ &= 2ad + 2af - (a-e) - (n+2)f - d(a-e) - f(a-g) + 2a \\ &\geq ad + af + g + de + fg + e > 0 \end{aligned}$$

Case 2: $b > 0$

$\left\lceil \frac{N}{n} \right\rceil = a + \left\lceil \frac{a+b}{n} \right\rceil; \left\lceil \frac{N}{n+1} \right\rceil = a + 1; \left\lceil \frac{N}{n+2} \right\rceil = a - \left\lfloor \frac{a-b}{n+2} \right\rfloor$

By definition, $b < n+1 \Rightarrow b - a < n+1$. Therefore, when $a - b < 0$, $\left\lfloor \frac{N}{n+2} \right\rfloor = a + 1$.

Case 2.1: $2 \leq a + b \leq n$ **and** $a - b < 0$

$\left\lceil \frac{N}{n} \right\rceil = a + 1; \left\lceil \frac{N}{n+1} \right\rceil = a + 1; \left\lceil \frac{N}{n+2} \right\rceil = a + 1$

$$L - R = 0$$

Case 2.2: $2 \leq a + b \leq n$ **and** $a - b \geq 0 \Rightarrow a - b < n$

$\lceil \frac{N}{n} \rceil = a+1; \lceil \frac{N}{n+1} \rceil = a+1; \lceil \frac{N}{n+2} \rceil = a$

$$\begin{aligned} L &= 2N(a+1) + 2Na + n(a+1) + (n+2)a + 2(n+1)(a+1)^2 \\ R &= 4N(a+1) + 2(n+1)(a+1) + n(a+1)^2 + (n+2)a^2 \\ L - R &= -2N + 2na + 4a \\ &= -2(an + a + b) + 2na + 4a = 2(a-b) \geq 0 \end{aligned}$$

Case 2.3: $a + b > n$ **and** $a - b < 0$

Let $a + b = dn + e; d \geq 1, 0 \leq e < n, d + e \geq 2$

$\lceil \frac{N}{n} \rceil = a + d + \lceil \frac{e}{n} \rceil ; \lceil \frac{N}{n+1} \rceil = a+1; \lceil \frac{N}{n+2} \rceil = a+1$

Case 2.3.1: $e = 0 \Rightarrow d \geq 2$

$\lceil \frac{N}{n} \rceil = a+d; \lceil \frac{N}{n+1} \rceil = a+1; \lceil \frac{N}{n+2} \rceil = a+1$

$$\begin{aligned} L &= 2N(a+d) + 2N(a+1) + n(a+d) + (n+2)(a+1) + 2(n+1)(a+1)^2 \\ R &= 4N(a+1) + 2(n+1)(a+1) + n(a+d)^2 + (n+2)(a+1)^2 \\ L - R &= 2N(d-1) + n(d-1) - 2an(d-1) - nd^2 + n \\ &= 2(an + a + b)(d-1) + n(d-1) - 2an(d-1) - n(d-1)(d+1) \\ &= (2a + 2b - nd)(d-1) > 0 \end{aligned}$$

Case 2.3.2: $e > 0$

$\lceil \frac{N}{n} \rceil = a+d+1; \lceil \frac{N}{n+1} \rceil = a+1; \lceil \frac{N}{n+2} \rceil = a+1$

$$\begin{aligned} L &= 2N(a+d+1) + 2N(a+1) + n(a+d+1) + (n+2)(a+1) + 2(n+1)(a+1)^2 \\ R &= 4N(a+1) + 2(n+1)(a+1) + n(a+d+1)^2 + (n+2)(a+1)^2 \\ L - R &= 2Nd - 2and - nd^2 - nd = 2d(an + a + b) - 2and - nd(d+1) \\ &= 2d(a+b) - (a+b-e)(d+1) = (a+b)(d-1) + e(d+1) > 0 \end{aligned}$$

Case 2.4: $a + b > n$ **and** $0 \leq a - b < n + 2$.

Let $a + b = dn + e$ where $d \geq 1, 0 \leq e < n$, and $d + e \geq 2$.

$\lceil \frac{N}{n} \rceil = a + d + \lceil \frac{e}{n} \rceil ; \lceil \frac{N}{n+1} \rceil = a+1; \lceil \frac{N}{n+2} \rceil = a$

Case 2.4.1: $e = 0 \Rightarrow d \geq 2$.

$\lceil \frac{N}{n} \rceil = a+d; \left\lceil \frac{N}{n+1} \right\rceil = a+1; \left\lceil \frac{N}{n+2} \right\rceil = a$

$$\begin{aligned}
L &= 2N(a+d) + 2Na + n(a+d) + (n+2)a + 2(n+1)(a+1)^2 \\
R &= 4N(a+1) + 2(n+1)(a+1) + n(a+d)^2 + (n+2)a^2 \\
L-R &= 2N(d-2) + nd - 2an(d-2) - nd^2 + 4a \\
&= 2(an+a+b)(d-2) + nd - 2an(d-2) - nd^2 + 4a \\
&= 2(a+b)(d-2) - nd(d-1) + 4a = 2(a+b)(d-2) - (a+b)(d-1) + 4a \\
&= (a+b)d + (a-3b) = (a+b)(d-1) + 2(a-b) > 0
\end{aligned}$$

Case 2.4.2: $e > 0$

$\lceil \frac{N}{n} \rceil = a+d+1; \left\lceil \frac{N}{n+1} \right\rceil = a+1; \left\lceil \frac{N}{n+2} \right\rceil = a$

$$\begin{aligned}
L &= 2N(a+d+1) + 2Na + n(a+d+1) + (n+2)a + 2(n+1)(a+1)^2 \\
R &= 4N(a+1) + 2(n+1)(a+1) + n(a+d+1)^2 + (n+2)a^2 \\
L-R &= 2N(d-1) + nd - 2an(d-1) - 2nd - nd^2 + 4a \\
&= 2(an+a+b)(d-1) - nd - 2an(d-1) - nd^2 + 4a \\
&= (a+b)(d-1) + 2(a-b) + e(d+1) > 0
\end{aligned}$$

Case 2.5: $a - b \geq n+2 \Rightarrow a+b > n$

Let $a+b = dn+e$, $d \geq 1, 0 \leq e < n, e+d \geq 2$

Let $a-b = f(n+2)+g$, $f \geq 1, 0 \leq g < n+2, f+g \geq 2$

$\lceil \frac{N}{n} \rceil = a+d+\lceil \frac{e}{n} \rceil; \left\lceil \frac{N}{n+1} \right\rceil = a+1; \left\lceil \frac{N}{n+2} \right\rceil = a-f$

Case 2.5.1: $e = 0 \Rightarrow d \geq 2$

$\lceil \frac{N}{n} \rceil = a+d; \left\lceil \frac{N}{n+1} \right\rceil = a+1; \left\lceil \frac{N}{n+2} \right\rceil = a-f$

$$\begin{aligned}
L &= 2N(a+d) + 2N(a-f) + n(a+d) + (n+2)(a-f) + 2(n+1)(a+1)^2 \\
R &= 4N(a+1) + 2(n+1)(a+1) + n(a+d)^2 + (n+2)(a-f)^2 \\
L-R &= 2Nd - 4N - 2Nf + nd - nf - 2f - 2nad - nd^2 + 2anf - nf^2 + 4af - 2f^2 + 4an + 4a \\
&= 2d(an+a+b) - 4(an+a+b) - 2f(an+a+b) + nd - nf - 2f \\
&\quad - 2nad - nd^2 + 2anf - nf^2 + 4af - 2f^2 + 4an + 4a \\
&= ad + b(d-2) + f(a-b) + g + fg > 0
\end{aligned}$$

Case 2.5.2: $e > 0$

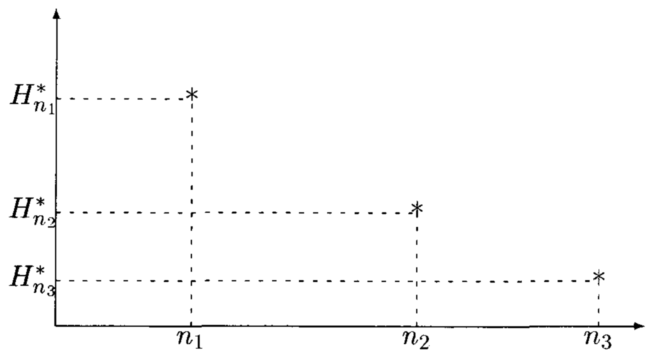


Figure 3.3: H^*_n vs. n

$\lceil \frac{N}{n} \rceil = a + d + 1; \lceil \frac{N}{n+1} \rceil = a + 1; \lceil \frac{N}{n+2} \rceil = a - f.$

$$\begin{aligned}
L &= 2N(a+d+1) + 2N(a-f) + n(a+d+1) + (n+2)(a-f) + 2(n+1)(a+1)^2 \\
R &= 4N(a+1) + 2(n+1)(a+1) + n(a+d+1)^2 + (n+2)(a-f)^2 \\
L - R &= 2Nd - 2N - 2Nf - nd - nf - 2f - 2nad - nd^2 + 2anf - nf^2 + 4af - 2f^2 + 2an + 4a \\
&= 2d(an+a+b) - 2(an+a+b) - 2f(an+a+b) - nd - nf \\
&\quad - 2f - 2nad - nd^2 + 2anf - nf^2 + 4af - 2f^2 + 2an + 4a \\
&= (ad-b) + b(d-1) + f(a-b) + g + fg + e + de > 0
\end{aligned}$$

Therefore, $(H^*_n - H^*_{n+1}) - (H^*_{n+1} - H^*_{n+2}) \geq 0$ for all $1 \leq n \leq N-2$. Thus, the marginal savings is non-increasing in n. □

Lemma 3.5 *H^*_n is a discrete convex function of n.*

Proof: Figure 3.3 shows the total holding cost H^*_n which satisfies Lemmas 3.3 and 3.4 at $n = n_1$, $n = n_3$ and $n = n_2 = [\alpha n_1 + (1-\alpha)n_3]$, where $0 < \alpha < 1$ and n_2 is the integer part of $\alpha n_1 + (1-\alpha)n_3$. By Lemma 3.4,

$$\frac{H^*_{n_1} - H^*_{n_2}}{n_2 - n_1} \geq \frac{H^*_{n_2} - H^*_{n_3}}{n_3 - n_2} \Longrightarrow \frac{H^*_{n_1} - H^*_{n_2}}{H^*_{n_2} - H^*_{n_3}} \geq \frac{n_2 - n_1}{n_3 - n_2} = \frac{(1-\alpha)}{\alpha}.$$

This implies

$$\alpha H^*_{n_1} + (1-\alpha)H^*_{n_3} \geq H^*_{n_2} = H^*_{(\alpha n_1 + (1-\alpha)n_3)}. \qquad □$$

Corollary 3.1 *The optimal total cost function is discrete convex in n.*

Proof: H_n^* is discrete convex in n. Since the delivery cost is linear in n and the sum of convex functions is convex again, the corollary follows. □

Since the optimal total cost function is discrete convex in n, we can find the optimal number of batches using binary search. The marginal savings in the holding cost at $n = r$ is $H_r^* - H_{r+1}^*$. Since the marginal savings in the holding cost is non-increasing and increasing the number of batches by 1 increases the delivery cost by d, there will not be any savings in the total cost by a further increase in the number of batches if $H_r^* - H_{r+1}^* \leq d$. Thus at a given r, we can check the marginal savings and decide which way to move in the binary search. Based on this argument, we give our algorithm *BatchNum* (Algorithm 3.1) for finding the optimal number of batches.

Algorithm 3.1 (Algorithm *BatchNum* for Optimal Number of Batches)

```
Step 1    Set $s = 1$, $l = N - 1$
Step 2    Calculate $\Delta_l = H_l^* - H_{l+1}^*$
          If $\Delta_l \geq d$ then set $n^* = l + 1$; Stop
          Calculate $\Delta_s = H_s^* - H_{s+1}^*$
          If $\Delta_s \leq d$ then set $n^* = s$; Stop
Step 3    Set $m = \lfloor \frac{s+l}{2} \rfloor$
          If $m = s$ then set $n^* = m + 1$; Stop
          Calculate $\Delta_m = H_m^* - H_{m+1}^*$
          If $\Delta_m = d$ then set $n^* = m$; Stop
          If $\Delta_m < d$ then set $l = m$; Go to Step 3
          Set $s = m$; Go to Step 3
```

Since the algorithm uses binary search over n, and we clearly have $n \leq N$, the complexity of *BatchNum* is $O(logN)$. Thus we have proved the following theorem.

Theorem 3.1 *Algorithm BatchNum finds an optimal number of batches for the single-product batch scheduling problem of the single-customer model in $O(logN)$ time. Thus the single-product single-customer batching problem can also be solved in $O(logN)$ time.*

Remark 3.1 Once we have found the optimal n, the AEBS policy and the formulas in Observation 4 clearly define the optimal batch sizes. Thus Theorem 3.1 leads to the interesting situation that we can find and describe the solution for the problem in $O(logN)$ time, which is polynomial in the size of the input for the problem, however, if we wanted to output explicitly the batch sizes $b_1, b_2, ..., b_n$ and the resulting schedule, this may require exponential time in the size of the input, since n itself may be exponentially sized.

Remark 3.2 It is easy to see that Algorithm *BatchNum* also gives the optimal number of batches when the machine needs a setup at the start of the first batch: In this case, each item will be delayed by s time units, thus the total cost will have an additional term wsN, which does not vary with n or the batch sizes, and thus the same n and the batch sizes will remain optimal as without setups.

3.3 One Product per Customer to Multiple Customers Problem

In this section, we analyse the supplier's problem when each customer requires a single product, but different customers may require different products. This model is applicable to the plastic manufacturer example discussed in Chapter 1. The model also applies to situations where a customer may need more than one product as long as different products are delivered in separate batches. The supplier needs a setup

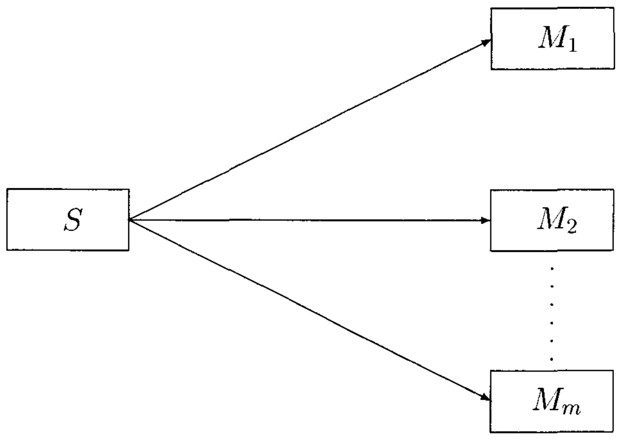


Figure 3.4: Multiple Customer Model.

s_i on the machine when switching the machine to produce product P_i succeeding a different product. Figure 4.6 illustrates a network representation of the model when the supplier delivers to m customers $M_1, M_2, ..., M_m$. We utilize the results obtained in the previous section to solve the supplier's general problem. We assume that orders to different customers are delivered in separate batches.

Lemma 3.6 *There exists an optimal schedule in which batches of the same product are processed consecutively.*

Proof: Consider an optimal schedule in which batches of the same product are not processed consecutively. Consider a part of the schedule in which the kth batch b_{ik} of product P_i starts at time t followed by r batches of products other than P_i and then the $(k+1)$th batch $b_{i,k+1}$ of product P_i as shown in Figure 3.5. (Since the holding cost of an item is independent of the customer, we have removed the subscript of the customer for simplicity.) As it is not important which product (only that other than P_i) the batches between b_{ik} and $b_{i,k+1}$ belong to, let us denote these r batches by $b_1^{'}, b_2^{'}, ..., b_r^{'}$ and the corresponding unit processing times, unit holding costs and setup

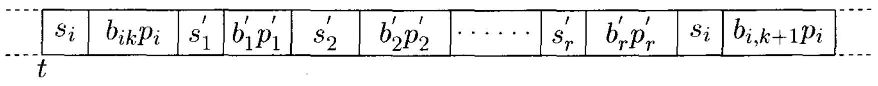


Figure 3.5: A Partial Batching Schedule

times by p_j', w_j' and s_j', respectively for $j = 1, 2, ..., r$. Note that s_j' is zero if b_{j-1}' and b_j' are batches of the same product.

Let the holding cost of b_{ik} and $b_{i,k+1}$ in this partial schedule be I_k and I_{k+1}, respectively. Also let the holding cost of b_j' be A_j for $j = 1, 2, ..., r$, and the total holding cost of the remaining batches be A. Then

$$I_k = b_{ik}w_i(t + s_i + b_{ik}p_i)$$
$$A_j = b_j'w_j'(t + s_i + s_1' + s_2' + ... + s_j' + b_{ik}p_i + b_1'p_1' + b_2'p_2' + ... + b_j'p_j'), \text{ for } j = 1, 2, ..., r$$
$$I_{k+1} = b_{i,k+1}w_i(t + 2s_i + s_1' + s_2' + ... + s_r' + b_{ik}p_i + b_1'p_1' + b_2'p_2' + ... + b_r'p_r' + b_{i,k+1}p_i)$$

Now consider the schedule in which $b_{i,k+1}$ is moved right after b_{ik} and let the corresponding holding costs for b_{ik} and $b_{i,k+1}$ be I_k' and I_{k+1}', respectively, and for b_j' let it be A_j', for $j = 1, 2, ..., r$. It is clear that the total holding cost of the remaining batches will not be worse than A. We also have

$$I_k' = b_{ik}w_i(t + s_i + b_{ik}p_i)$$
$$A_j' = b_j'w_j'(t + s_i + s_1' + s_2' + ... + s_j' + b_{ik}p_i + b_{i,k+1}p_i + b_1'p_1' + b_2'p_2' + ... + b_j'p_j'), \text{ for } j = 1, 2, ..., r$$
$$I_{k+1}' = b_{i,k+1}w_i(t + s_i + b_{ik}p_i + b_{i,k+1}p_i)$$

Let $H' = (I_k' + I_{k+1}' + A_1' + A_2' + ... + A_r')$ and $H = (I_k + I_{k+1} + A_1 + A_2 + ... + A_r)$. Then

$$H' - H = b_{i,k+1}p_i(b_1'w_1' + b_2'w_2' + ... + b_r'w_r') - b_{i,k+1}w_i(s_i + s_1' + s_2' + ... + s_r' + b_1'p_1' + b_2'p_2' + ... + b_r'p_r').$$

Now consider another schedule derived from the original schedule by moving b_{ik} just before $b_{i,k+1}$ and let the corresponding holding costs for b_{ik} and $b_{i,k+1}$ be $I_k^{"}$ and $I_{k+1}^{"}$, respectively, and for b_j' let it be $A_j^{"}$, for $j = 1, 2, ..., r$. Again the holding cost of the remaining batches will not be worse than A. We have

$$I_k^{"} = b_{ik}w_i(t + s_1' + s_2' + ... + s_r' + s_i + b_1'p_1' + b_2'p_2' + ... + b_r'p_r' + b_{ik}p_i)$$
$$A_j^{"} = b_j'w_j'(t + s_1' + s_2' + ... + s_j' + b_1'p_1' + b_2'p_2' + ... + b_j'p_j'), \text{ for } j = 1, 2, ..., r$$
$$I_{k+1}^{"} = b_{i,k+1}w_i(t + s_1' + s_2' + ... + s_r' + s_i + b_1'p_1' + b_2'p_2' + ... + b_r'p_r' + b_{ik}p_i + b_{i,k+1}p_i)$$

Let $H^{"} = (I_k^{"} + I_{k+1}^{"} + A_1^{"} + A_2^{"} + ... + A_r^{"})$. Then we get

$$\begin{aligned} H^{"} - H \;=\; & b_{ik}w_i(s_1' + s_2' + ... + s_r' + b_1'p_1' + b_2'p_2' + ... + b_r'p_r') \\ & -b_{ik}p_i(b_1'w_1' + b_2'w_2' + ... + b_r'w_r') \\ & -s_i(b_1'w_1' + b_2'w_2' + ... + b_r'w_r') - b_{i,k+1}w_is_i \end{aligned}$$

Since H' and $H^{"}$ are obtained by moving batches of the optimal schedule, we know that $H' \geq H$ and $H^{"} \geq H$. Let us assume now that $H' > H$. Then,

$$p_i(b_1'w_1' + b_2'w_2' + ... + b_r'w_r') > w_i(s_i + s_1' + s_2' + ... + s_r' + b_1'p_1' + b_2'p_2' + ... + b_r'p_r'),$$

which implies

$b_{ik}p_i(b_1'w_1' + b_2'w_2' + ... + b_r'w_r') + s_i(b_1'w_1' + b_2'w_2' + ... + b_r'w_r') + b_{i,k+1}w_is_i > b_{ik}w_i(s_1' + s_2' + ... + s_r' + b_1'p_1' + b_2'p_2' + ... + b_r'p_r')$.

The last inequality, however, is equivalent to $H^{"} - H < 0$. This contradicts the assumption that the partial schedule is from the optimal schedule. So we could not have $H' > H$ originally, i.e. $H' = H$.

Repeatedly using the above argument, we can construct an alternative optimal schedule in which batches of every product are processed consecutively, thus proving the lemma. □

Lemma 3.7 *There exists an optimal schedule in which batches of the same product to the same customer are processed consecutively.*

Proof: Let us consider an optimal schedule satisfying Lemma 3.6, that is in which batches of the same product are consecutive. If this schedule does not satisfy the current lemma, then there will be two batches $b_{jk}^{(l)}$ and $b_{j,k+1}^{(l)}$, for customer M_l, which are not consecutive, but every other batch between them is for product P_j. We can apply Property 3.1 to the part of the schedule for P_j, which implies that moving $b_{j,k+1}^{(l)}$ next to $b_{jk}^{(l)}$ will not affect the total holding cost. Repeatedly using this argument, we can obtain a schedule satisfying Lemma 3.7. $\square$

Lemma 3.6 allows us to make the batching decisions for a given product separately from other products. Since delivery to each customer is done separately in the optimal schedule and we can assume that batches of the same product to the same customer are processed consecutively, optimal batch sizes of a given product to a given customer are also independent of the batching of the same product to other customers. Therefore, the optimal number of batches and the resulting batch sizes for product P_j to customer $M^{(i)}$ can be obtained by using the algorithm BatchNum with input $N_j^{(i)}$, where $N_j^{(i)}$ is the number of items of product P_j to be delivered to customer $M^{(i)}$.

Lemma 3.8 *In the optimal schedule, products are scheduled in non-decreasing order of* $\frac{s_i+p_iN_i}{w_iN_i}$, *where* $N_i = \sum_{k=1}^{m} N_i^{(k)}$.

Proof: Consider an optimal schedule in which, contrary to the lemma, product P_i immediately precedes product P_j, where $\frac{s_i+p_iN_i}{w_iN_i} > \frac{s_j+p_jN_j}{w_jN_j}$, and processing of product P_i starts at time t as shown in Figure 3.6. In the figure, we have removed the subscript for customers since the holding cost does not depend on the customer. In total, there are n_i batches of P_i and n_j batches of P_j.

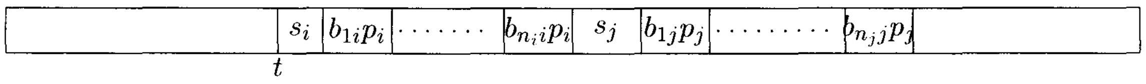


Figure 3.6: A Schedule for Lemma 3.8

Let H_i and H_j be the holding cost of product P_i and P_j, respectively, in the schedule. Then

$$H_i = w_i\Big((t+s_i)N_i + \sum_{r=1}^{n_i} b_{ir}\sum_{s=1}^{r} p_i b_{is}\Big)$$
$$H_j = w_j\Big((t+s_i+s_j+p_iN_i)N_j + \sum_{r=1}^{n_j} b_{jr}\sum_{s=1}^{r} p_j b_{js}\Big)$$

Let us exchange the batches of P_j with the batches of P_i without changing their sizes or relative order. Let the corresponding holding costs be $H_j^{'}$ and $H_i^{'}$ for P_j and P_i in the resulting schedule. Then

$$H_i^{'} = w_i\Big((t+s_i+s_j+p_jN_j)N_i + \sum_{r=1}^{n_i} b_{ir}\sum_{s=1}^{r} p_i b_{is}\Big)$$
$$H_j^{'} = w_j\Big((t+s_j)N_j + \sum_{r=1}^{n_j} b_{jr}\sum_{s=1}^{r} p_j b_{js}\Big)$$
$$(H_i^{'}+H_j^{'}) - (H_i+H_j) = w_iN_i(s_j+p_jN_j) - w_jN_j(s_i+p_iN_i)$$

Since $\frac{s_j+p_jN_j}{w_jN_j} < \frac{s_i+p_iN_i}{w_iN_i}$, we have $(H_i^{'}+H_j^{'}) - (H_i+H_j) < 0$. This contradicts the optimality assumption for the original schedule. □

Therefore, the optimal product sequence can be obtained by Lemma 3.8. The optimal batching of a product P_i needs at most $O(\sum_{k=1}^{m} \log N_i^{(k)})$ calculations by Theorem 3.1. Therefore, the total complexity of our solution for the multi-customer multi-product batching problem is $O(\sum_{k=1}^{m}\sum_{i=1}^{S} \log N_i^{(k)}) \leq O(SmlogN_{max})$, where S is the number of product types, and $N_{max} = \max_{k=1,2,\ldots,m,\, i=1,2\ldots,S} N_i^{(k)}$. Thus we have proved the following theorem.

Theorem 3.2 *There exists a polynomial-time algorithm which solves the multi-customer multi-product optimal batch scheduling problem in $O(mSlogN_{max})$ time under the assumption that different products are delivered in separate batches.*

We observe that in practice suppliers often use long production runs making enough to satisfy the demand for a product over the entire scheduling period and thus saving the avoidable setups. Lemma 3.6 means that this "intuitive" solution is optimal in the current model. It also means that when looking for the optimal batch sequence, we can assume that if there is more than one batch of a product for the same customer, then these batches are scheduled consecutively. We can use this to derive the optimal batch sizes and sequence, but note that reordering these batches of identical products after, does not affect our objective function. Thus once the optimal batch sizes have been determined, the supplier is free to use *any* sequence of the batches of identical product. For example, the supplier can cyclically deliver one batch of a product to every customer before delivering any subsequent batches of the same product to the same customer without increasing the total cost.

So far, we have analyzed batch scheduling at the supplier with the assumption that different products are never batched together. In the next sections we study polynomially solvable cases of the supplier's problem without this assumption.

3.4 Polynomially Solvable Special Cases of the Supplier's General Problem

The supplier's general problem was proven to be strongly NP-hard [19] when different products can be batched together even if we have only a single customer. In the next subsections we study special cases of this problem which are solvable in polynomial time.

3.4.1 Batching of Jobs with Fixed Job Sequence

We prove that batch scheduling problems of multiple products (jobs) at the supplier with a single customer can be solved polynomially when the jobs follow an arbitrary but fixed job sequence. Coffman et al. [14] study the batch scheduling problem to minimize the sum of the completion times for jobs with arbitrary processing times when there is a constant setup time at the start of each batch. They present an $O(N)$ algorithm for the optimal batching of any job sequence of N jobs. They further prove that in the optimal schedule, jobs are ordered in SPT order to minimize the sum of completion times. Since the SPT job sequence can be obtained in $O(NlogN)$ time, they are able to solve the optimal batch scheduling problem in $O(NlogN)$ time. Later, Albers and Brucker [1] extended this study and proved that the optimal batching of jobs with identical processing times and arbitrary weights can be solved in $O(NlogN)$ time, and the optimal batching of jobs with arbitrary processing times and arbitrary weights with any *given* job sequence can also be solved in $O(N)$ time. In this section, we study the batching of jobs with arbitrary processing times and arbitrary weights of any fixed job sequence when there is an associated batch delivery cost. We prove that this problem can be solved polynomially by an extension of the algorithm of Coffman et al. with some modifications, similar to the ones used by Albers and Brucker. We follow the notation of Albers and Brucker.

Without loss of generality let us assume that the fixed job sequence we want to batch optimally is $J_1, J_2, \ldots, J_N$. Consider a batching solution of this job sequence with k batches

$$BS = \{J_{i_1}, \ldots, J_{i_2-1}\}, \{J_{i_2}, \ldots, J_{i_3-1}\}, \ldots, \{J_{i_{k-1}}, \ldots, J_{i_k-1}\}, \{J_{i_k}, \ldots, J_N\},$$

where i_j is the index of the first job in the jth batch and $1 = i_1 < i_2 < \cdots < i_k \leq N$.

We show that this problem can be reduced to the special shortest path problem discussed by Albers and Brucker.

Let $F(BS)$ be the total cost (sum of the weighted flow time and the delivery cost) of the batching solution BS. Since the completion time of every item in batch $\{J_{i_j}, J_{i_j+1}, \ldots, J_{i_{j+1}-1}\}$ is $\sum_{v=i_1}^{i_{j+1}-1} p_v$, $F(BS)$ can be calculated as follows:

$$F(BS) \;=\; \sum_{j=1}^{k}\Big(\Big(\sum_{v=i_j}^{N} w_v\Big)\Big(\sum_{v=i_j}^{i_{j+1}-1} p_v\Big) + d\Big),$$

where $i_{k+1} = N+1$ is a 'dummy' job.

If we set

$$c_{ij} = \Big(\sum_{v=i}^{N} w_v\Big)\Big(\sum_{v=i}^{j-1} p_v\Big) + d, \tag{3.1}$$

which is the cost contribution by a batch containing jobs $J_i, J_{i+1}, \ldots, J_{j-1}$, then $F(BS) = \sum_{j=1}^{k} c_{i_j, i_{j+1}-1}$. Furthermore, for any $k > j$,

$$c_{ik} - c_{ij} = \Big(\sum_{v=i}^{N} w_v\Big)\Big(\sum_{v=j}^{k-1} p_v\Big) = f(i)h(j,k), \text{ where } f(i) = \sum_{v=i}^{N} w_v \text{ and } h(j,k) = \sum_{v=j}^{k-1} p_v$$

Since $f(i)$ is monotone nonincreasing and $h(j,k) > 0$ for all $j < k$, our problem can also be formulated as a special shortest path problem introduced by Albers and Brucker. The following is the corresponding network for our problem:

a. Each job J_i $(i = 1, 2, \ldots, N)$ is represented by a vertex i in the network.

b. Since the job sequence $J_1, J_2, \ldots, J_N$ is given, the network contains edges (i,j) only if $i < j$.

c. Any edge (i,j) is assigned an edge length c_{ij}.

d. A dummy job J_{N+1} with $p_{N+1} = w_{N+1} = 0$ is added.

tail	$\cdots$	*next(i)*	*i*	*previous(i)*	$\cdots$	*head*

Figure 3.7: Structure of a queue q.

The batching problem then is equivalent to finding the shortest path from vertex 1 to vertex $N+1$ in the network.

Let F_j be the length of the shortest path from vertex j to vertex $N+1$, and $F_j(k)$ be the length of a shortest path from j to $N+1$ which contains (j,k) as first edge. Then,

$F_j(k) = c_{jk} + F_k$, and $F_j = min\{F_j(k) | j < k \leq N+1\}$.

Therefore, $F_j(k) \leq F_j(l)$ for vertices $j < k < l$ is equivalent to

$$F_j(k) - F_j(l) \;\; = \;\; c_{jk} + F_k - c_{jl} - F_l \leq 0$$

But $c_{jl} - c_{jk} = f(j)h(k,l)$, therefore the condition can be written as $f(j) \geq \frac{F_k - F_l}{h(k,l)}$.

Thus for any two vertices k, l $(k < l)$, if the *threshold* $\delta(k,l) = \frac{F_k - F_l}{h(k,l)} \leq f(j)$, then $F_j(k) \leq F_j(l)$ and l is called *not better* than k with respect to j. On the other hand, if $f(j) < \delta(k,l)$, then $F_j(k) > F_j(l)$ and l is called *better* than k with respect to j.

Therefore, the problem can be solved in $O(N)$ time using a slightly modified version of the algorithm by Albers and Brucker. They use the queue data structure shown in Figure 3.7 to solve the problem.

The correctness of the algorithm is based on the following two lemmas, which can be proved by the same proof as in Albers and Brucker [1].

Lemma 3.9 *Let $1 \leq j < k < l$ be vertices satisfying $f(j) \geq \delta(k,l)$. Then $F_i(k) \leq F_i(l)$, for all $i = 1, 2, ..., j$.*

Algorithm 3.2 (Modified Algorithm of Albers and Brucker)

```
        begin
Step 1  $q = N + 1$; $F_{N+1} = 0$
Step 2: for $j = N$ to 1 do begin
Step 3:   while $head(q) \neq tail(q)$ and $f(j) \geq \delta(next(head(q)), head(q))$ do
            Delete $head(q)$ from $q$;
Step 4:   $N(j) = head(q)$; $F_j = c_{j,N(j)} + F_{N(j)}$
Step 5:   while $head(q) \neq tail(q)$ and $\delta(j, tail(q)) \leq \delta(tail(q), previous(tail(q)))$ do
            Delete $tail(q)$ from $q$
Step 6:   Add $j$ to the tail of $q$;
        end
        end
```

Lemma 3.10 *Suppose $\delta(j,k) \leq \delta(k,l)$ for some vertices $1 \leq j < k < l \leq n+1$. Then for each vertex i, $1 \leq i \leq j$, either k is not better than j or l is better than k with respect to i.*

In order to obain the linear time complexity, we need to do some preprocessing. Each c_{ij} value needed can be calculated in $O(1)$ time by equation (3.1). The $f(j)$ values can easily be computed in a preprocessing step in $O(N)$ time. To make an $h(j,k)$ value computable in $O(1)$ time, we also compute in the preprocessing step the following:

$$sp(i) = \begin{cases} 0 & \text{for } i = 1 \\ \sum_{v=1}^{i-1} p_v & \text{for } i = 2, 3, \ldots, N+1. \end{cases}$$

This clearly requires $O(N)$ time and using $h(j,k) = sp(k) - sp(j)$ makes any $h(j,k)$ computable in $O(1)$ time. Finally, the algorithm requires at most $O(N)$

iterations, as each vertex gets added or deleted at most once from q. Thus we have the following theorem.

Theorem 3.3 *Algorithm 3.2 computes the optimal batching of a given job sequence in $O(N)$ time.*

The following example illustrates the detailed execution of the algorithm.

Example 3.1: Consider optimal batching of a job sequence $(J_1, J_2, J_3, J_4, J_5, J_6)$. Let $p_1 = p_2 = 10$; $p_3 = p_4 = 100$; $p_5 = p_6 = 20$; $w_1 = w_2 = 50$; $w_3 = w_4 = 60$; $w_5 = w_6 = 11$; $d = 1200$.

Initialization: $q = \{7\}$; $head(q) = 7$; $tail(q) = 7$; $F_7 = 0$.

$j = 6$ $N(6) = 7$; $F_6 = 1420$
Add 6 to q; $q = \{6, 7\}$; $tail(q) = 6$

$j = 5$ $f(5) = 22$; $next(head(q)) = 6$
$\delta(6, 7) = 71 > f(5)$
$N(5) = 7$; $F_5 = 2080$
$\delta(5, 6) = 33$; $previous(tail(q)) = 7$
$\delta(6, 7) = 71 > \delta(5, 6)$; Remove 6 from q
Add 5 to q; $q = \{5, 7\}$; $tail(q) = 5$

$j = 4$ $f(4) = 82$; $next(head(q)) = 5$
$\delta(5, 7) = 70 < f(4)$; Remove 7 from q
$q = \{5\}$; $head(q) = 5$
$N(4) = 5$; $F_4 = 11480$
Add 4 to q; $q = \{4, 5\}$; $tail(q) = 4$

$j = 3$ $f(3) = 142$; $next(head(q)) = 4$
$\delta(4,5) = 94 < f(3)$; Remove 5 from q
$q = \{4\}$; $head(q) = 4$
$N(3) = 4$; $F_3 = 26880$
Add 3 to q; $q = \{3,4\}$; $tail(q) = 3$

$j = 2$ $f(2) = 192$; $next(head(q)) = 3$
$\delta(3,4) = 154 < f(2)$; Remove 4 from q
$q = \{3\}$; $head(q) = 3$
$N(2) = 3$; $F_2 = 30000$
Add 2 to q; $q = \{2,3\}$; $tail(q) = 2$

$j = 1$ $f(1) = 242$; $next(head(q)) = 2$
$\delta(2,3) = 312$
$N(1) = 3$; $F_1 = 32920$

j	6	5	4	3	2	1
$N(j)$	7	7	5	4	3	3

Therefore, the optimal batching is (J_1, J_2) (J_3) (J_4) (J_5, J_6) with an objective value of 32,920.

We have proved that there exists a polynomial time algorithm when the job sequence is given. Therefore, when there are multiple products to be batched, if job sequencing and batching are separable and if the optimal job sequence can be found in polynomial time then the batch scheduling problem can be solved in polynomial time. We use this property to solve special cases of the batch scheduling problem optimally in the following two subsections.

3.4.2 Batching of Jobs with Identical Weights

In batching of jobs with identical weights all the jobs have the same weight, i. e., $w_i = w$ for $i = 1, 2, \ldots, N$. This model is applicable to the pasta manufacturer example discussed in Chapter 1. This problem is equivalent to the batching problem to minimize the sum of the completion times and the delivery costs.

Property 3.2 *There exists an optimal solution in which jobs are sequenced in SPT order.*

Proof: If job J_j precedes job J_i in a schedule, where $p_j > p_i$, then interchanging jobs J_j and J_i will not increase the total holding cost without affecting the number of batches. $\square$

Corollary 3.2 *The optimal batch scheduling problem of jobs with identical weights can be solved in $O(NlogN)$ time.*

Proof: From Property 3.2, in the optimal schedule, jobs are sequenced in SPT order. SPT order can be obtained in $O(NlogN)$ time. Further, the optimal batching of this given job sequence can be done in $O(N)$ time using Algorithm 3.2. Thus, the complexity of the problem is $O(NlogN)$. $\square$

Corollary 3.2 improves the complexity of the solution from $O(N^2)$, which is the complexity of the algorithm given for the same problem by Hall and Potts [19]. They provide a dynamic programming algorithm to find optimal batch sizes for given job sequences to minimize the sum of flow time and delivery costs with time complexity $O(N^{m+1})$, where N is the total number of jobs delivered to m customers.

3.4.3 Batching of Jobs with Identical Processing Times

We study batch scheduling of jobs with identical processing time, i.e., $p_i = p$ for $i = 1, 2, \ldots, N$.

Property 3.3 *There exists an optimal solution in which jobs are scheduled in non-increasing order of job weights.*

Proof: If job J_j precedes J_i, where $w_j < w_i$, then interchanging J_j and J_i will not increase the total holding cost without changing the number of batches. □

Corollary 3.3 *The optimal batch scheduling of jobs with identical processing times can be solved in $O(NlogN)$ time.*

Proof: We need $O(NlogN)$ time to order the jobs in non-increasing weight order and $O(N)$ time to get the optimal batching for this given job sequence using Algorithm 3.2. □

3.5 Multiple Customer Batching of Jobs with Identical Processing Times

Here we prove that batch scheduling of jobs with identical processing times with multiple customers can be solved polynomially. We denote by $\sigma^{(k)}$ the job sequence of customer M_k. In the batch scheduling problem for multiple customers, we have to find the job sequence of each customer, batch sizes for each customer, and the sequence of the batches.

Lemma 3.11 *There exists an optimal batch schedule in which jobs of each customer are sequenced in nondecreasing weight order.*

Proof: If the lemma is not true then there will be at least two jobs $J_i^{(k)}$ and $J_j^{(k)}$ of customer M_k such that $J_i^{(k)}$ follows $J_j^{(k)}$ in $\sigma^{(k)}$, where $w_j^k < w_i^{(k)}$. If we interchange $J_i^{(k)}$ and $J_j^{(k)}$, the holding cost will not increase since the batch completion time remains the same while the job with larger weight is as signed to an earlier position. □

Therefore, job sequencing and batching are separable for this problem. The algorithm by Hall and Potts [19] can be used to solve the batching problem of this fixed job sequence. Job sequencing requires $O(mlogN_{max})$ time, where $N_{max} = \max_{k=1,2,\ldots,m} N_k$ and N_k is the demand from customer M_k. The algorithm by Hall and Potts requires $O((\sum_{k=1}^{m} N_k)^{m+1})$ time to find the optimal batching.

Chapter 4

Approximation Algorithms for the Supplier's General Problem

4.1 Introduction

In this chapter, we study the batch scheduling of jobs with arbitrary processing times and arbitrary weights and provide a 2-approximation algorithm for this problem. The batch scheduling problem with arbitrary processing times and arbitrary weights is strongly NP-hard (Hall and Potts [19]). This justifies the development of an approximation algorithm. We first study some properties of preemptive batch scheduling and based on these properties, we develop our 2-approximation algorithm.

4.2 Multiple Products, Single Customer

In this section, we study the job sequencing and batching of multiple products for a single customer at the supplier.

Lemma 4.1 *If $w_i p_j \geq d$, (for all $i \neq j$, $i, j = 1, 2, \ldots, N$), then in the optimal batching solution, jobs are sequenced in WSPT order.*

Proof: When $w_i p_j \geq d$, (for all $i \neq j$ and $i, j = 1, 2, \ldots, N$), then each job is delivered as soon as its processing is completed. Thus the problem is equivalent to scheduling N jobs so that the weighted sum of the completion times is minimized. Smith [37] proves that WSPT job sequence is optimal for minimum weighted completion time problems. □

The example given below, however, proves that the WSPT job sequence may not provide the optimal batching solution for the general problem.

Example 4.2: Counter example for WSPT job sequence being optimal.

Consider the following example with three jobs and $d = 1200$:

J_i	J_1	J_2	J_3
p_i	10	100	20
w_i	50	60	11
$\frac{p_i}{w_i}$	0.20	1.7	1.8

The optimal batching solution for this problem is (J_1, J_3) in the first batch and (J_2) in the second batch with total cost of 12030. The optimal batching solution of the WSPT job sequence, however, is (J_1) in the first batch and (J_2, J_3) in the second batch with total cost 12130. Thus the total cost of the optimal batching of the WSPT job sequence is approximately 1.008 times the total cost of the optimal batching solution.

We first prove that the WSPT job sequence is optimal for a preemptive version of the problem. Based on this, we then prove that there exists a 2-approximation algorithm for the general problem.

Our development of a 2-approximation algorithm for the general batching problem is based on a strong lower bound for the optimum, which is derived from a

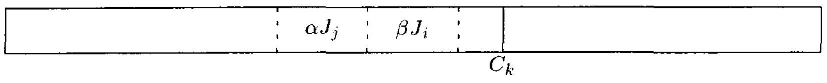


Figure 4.1: Schedule when αJ_j and βJ_i are assigned to B_k.

preemptive version of the problem. It is interesting to note that the algorithm does *not* have to solve the preemptive problem. We only use the properties of its optimal solution in bounding the approximation ratio of our algorithm.

4.2.1 Preemptive Batch Scheduling of Multiple Products to a Single Customer

In the *preemptive* batch scheduling problem, we assume that if a fraction α of a job J_i (denoted by αJ_i) is completed, that α fraction can be delivered separately from the other parts of J_i. In calculating the objective function, we assign weight αw_i and processing time αp_i to the fractional job αJ_i. Thus the processing-time-to-weight ratio of the fractional job αJ_i is the same as the ratio for J_i. The following lemmas describe useful properties for this preemptive problem.

Lemma 4.2 *There is an optimal schedule in which jobs are scheduled in WSPT order for the preemptive batching problem.*

Proof: Let us assume that the lemma does not hold. Then there exists an optimal batching in which a part β of job J_i (βJ_i) immediately follows a part α of job J_j (αJ_j) in the schedule, where $\frac{p_j}{w_j} > \frac{p_i}{w_i}$. Note that $0 < \alpha, \beta \leq 1$.

If αJ_j and βJ_i belong to the same batch B_k (refer to Figure 4.1), then moving βJ_i before αJ_j will not change the weighted flow time. If αJ_j and βJ_i are assigned to different batches B_k and B_{k+1} (refer to Figure 4.2), then moving βJ_i before αJ_j will change the total weighted flow time, and let this change be Δ. While exchanging

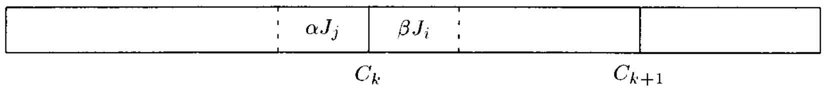


Figure 4.2: Schedule when αJ_j and βJ_i are assigned to B_k and B_{k+1}

the partial jobs, we *redefine* the boundary of B_k so that the batch completion times C_k and C_{k+1} will *not* change. Furthermore, the exchange of the jobs does not change the delivery cost.

Case 1 ($\alpha p_j \geq \beta p_i$): After moving the jobs, all of βJ_i and $(\alpha - \beta\frac{p_i}{p_j})J_j$ of αJ_j will be assigned to batch B_k and the remaining $\beta\frac{p_i}{p_j}$ part of αJ_j will be assigned to batch B_{k+1}.

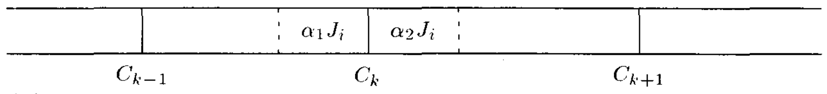
$$\begin{aligned}
\Delta &= -\beta w_i(C_{k+1} - C_k) + (\beta\frac{p_i}{p_j})w_j(C_{k+1} - C_k) \\
&= -\beta p_i(C_{k+1} - C_k)(\frac{w_i}{p_i} - \frac{w_j}{p_j}), \text{ but} \\
C_{k+1} - C_k &> 0 \text{ and } \frac{w_i}{p_i} - \frac{w_j}{p_j} > 0 \text{ by assumption. Therefore} \\
\Delta &< 0, \text{ contradicting the optimality of the original schedule.}
\end{aligned}$$

Case 2 ($\alpha p_j < \beta p_i$): After moving the jobs, $(\alpha\frac{p_j}{p_i})J_i$ of βJ_i will be assigned to B_k and $(\beta - \alpha\frac{p_j}{p_i})J_i$ of βJ_i and all of αJ_j will be assigned to B_{k+1}.

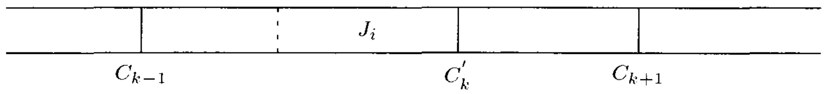
$$\begin{aligned}
\Delta &= -(\alpha\frac{p_j}{p_i})w_i(C_{k+1} - C_k) + \alpha w_j(C_{k+1} - C_k) \\
&= -\alpha p_j(C_{k+1} - C_k)(\frac{w_i}{p_i} - \frac{w_j}{p_j}) \\
\text{Therefore } \Delta &< 0, \text{ a contradiction again.}
\end{aligned}$$

Thus moving βJ_i before αJ_j will decrease the total weighted flow time, and the total cost will be reduced in each case. This, however, contradicts the optimality of the original schedule. Thus the optimal schedule must sequence the jobs in WSPT order. □

Remark 4.1 We note that Lemma 4.2 clearly implies that there is an optimal preemptive schedule in which fractional parts of the same job are sequenced consecutively.



(a) Partial schedule when $\alpha_1 J_i$ and $\alpha_2 J_i$ are assigned to B_k and B_{k+1}



(b) Partial schedule when rounded according to Lemma 4.4 Case 1.

Figure 4.3: A partial schedule of preemptive solution.

Lemma 4.3 *In the optimal schedule with preemption, if fractional jobs $\alpha_1 J_i$ and $\alpha_2 J_i$ of job J_i ($\alpha_1 + \alpha_2 \leq 1$ and $\alpha_1, \alpha_2 > 0$) are assigned to batches B_k and B_{k+1} as in Figure 4.3 (a), then the total processing time for B_{k+1}, T_{k+1}, and the total weight for B_k, W_k, satisfy $T_{k+1} = \frac{W_k}{\bar{w}_i}$, where $\bar{w}_i = \frac{w_i}{p_i}$.*

Proof:

Case 1: Move δJ_i ($0 < \delta \leq \alpha_1$) of $\alpha_1 J_i$ to B_{k+1}

Let the change in the weighted flow time be Δ_1.

$$\begin{aligned} \Delta_1 &= \delta w_i T_{k+1} - (W_k - \delta w_i)\delta p_i \\ \text{But } \Delta_1 &\geq 0 \text{ by optimality} \\ \text{Therefore, } \delta w_i T_{k+1} &\geq \delta p_i (W_k - \delta w_i) \\ w_i T_{k+1} &\geq p_i W_k - \delta w_i p_i \end{aligned}$$

Since the last inequality holds for arbitrarily small $\delta > 0$, we must have

$$\begin{aligned} w_i T_{k+1} &\geq p_i W_k \\ T_{k+1} &\geq \frac{W_k}{\bar{w}_i} \end{aligned} \tag{4.1}$$

Case 2: Move δJ_i ($0 < \delta \leq \alpha_2$) of $\alpha_2 J_i$ to B_k

Let the change in the weighted flow time be Δ_2.

$$\begin{aligned}\Delta_2 &= W_k\delta p_i - \delta w_i(T_{k+1} - \delta p_i)\\ \text{But } \Delta_2 &\geq 0, \text{ thus}\\ \delta w_i T_{k+1} &\leq \delta p_i(W_k + \delta w_i)\\ w_i T_{k+1} &\leq p_i W_k + \delta p_i w_i\end{aligned}$$

Since the last inequality must hold for any $\delta > 0$, we must have

$$\begin{aligned}w_i T_{k+1} &\leq p_i W_k\\ T_{k+1} &\leq \frac{W_k}{\bar{w}_i}\end{aligned} \tag{4.2}$$

From inequalities (4.1) and (4.2), $T_{k+1} = \frac{W_k}{\bar{w}_i}$. □

In the optimal preemptive batch schedule, a job J_i can be assigned to a single batch B_k, assigned over two batches B_k and B_{k+1}, or assigned over $r + 2$ $(r \geq 1)$ batches $B_k, B_{k+1}, \ldots, B_{k+r+1}$.

4.2.2 Non-preemptive Batch Scheduling of Multiple Product to a Single Customer

In this section, we use the properties of the preemptive solution and round the preempted jobs. The next three lemmas show how these solutions can be rounded into non-preemptive schedules and estimate the resulting errors.

Lemma 4.4 *Consider a job J_i which is split over two batches B_k and B_{k+1} with fractional jobs $\alpha_1 J_i$ and $\alpha_2 J_i$ $(\alpha_1 + \alpha_2 = 1)$ in the optimal preemptive schedule. Let us move the fractional job $\alpha_2 J_i$ (resp. $\alpha_1 J_i$) from B_{k+1} (resp. B_k) to B_k (resp. B_{k+1}) if $\alpha_1 \geq \alpha_2$ (resp. $\alpha_1 < \alpha_2$), and let Δ be the change in the weighted flow time. Then $\Delta \leq \frac{H_i}{3}$, where H_i is the holding cost of J_i in the preemptive optimal schedule.*

Proof: Refer to Figure 4.3 (a).

$$\begin{aligned} H_i &\geq (\alpha_1 w_i)(\alpha_1 p_i) + (\alpha_2 w_i)(\alpha_1 p_i + \alpha_2 p_i) \\ &\geq \alpha_1^2 w_i p_i + \alpha_2^2 w_i p_i + \alpha_1 \alpha_2 w_i p_i \\ &\geq 3\ min\{\alpha_1^2, \alpha_2^2\} w_i p_i \qquad (4.3) \end{aligned}$$

Case 1: $\alpha_1 \geq \alpha_2$ (refer to Figure 4.3 (b)).

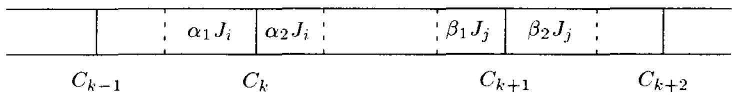
$$\begin{aligned} \Delta &= W_k \alpha_2 p_i - \alpha_2 w_i (T_{k+1} - \alpha_2 p_i) \\ &= W_k \alpha_2 p_i - \alpha_2 w_i T_{k+1} + \alpha_2 w_i (\alpha_2 p_i) \\ \text{From Lemma 4.3, } W_k \frac{p_i}{w_i} &= T_{k+1} \\ \text{Therefore, } \Delta &= (T_{k+1} \frac{w_i}{p_i}) \alpha_2 p_i - \alpha_2 w_i T_{k+1} + \alpha_2^2 w_i p_i \\ &= \alpha_2^2 w_i p_i \qquad (4.4) \end{aligned}$$

Case 2: $\alpha_1 < \alpha_2$

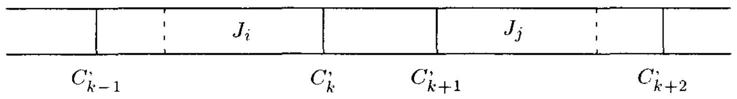
$$\begin{aligned} \Delta &= -(W_k - \alpha_1 w_i)\alpha_1 p_i + \alpha_1 w_i T_{k+1} \\ &= -W_k \alpha_1 p_i + \alpha_1 w_i T_{k+1} + \alpha_1 w_i (\alpha_1 p_i) \\ \text{From Lemma 4.3, } W_k \frac{p_i}{w_i} &= T_{k+1} \\ \text{Therefore, } \Delta &= (T_{k+1} \frac{w_i}{p_i}) \alpha_1 p_i - \alpha_1 w_i T_{k+1} + \alpha_1^2 w_i p_i \\ &= \alpha_1^2 w_i p_i \qquad (4.5) \end{aligned}$$

From equations (4.4) and (4.5), $\Delta \leq min\{\alpha_1^2, \alpha_2^2\} w_i p_i$. Therefore, from inequality (4.3), $\Delta \leq \frac{H_i}{3}$. □.

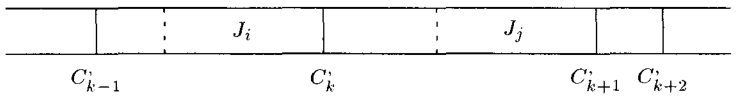
Lemma 4.5 *Consider two jobs J_i and J_j where J_i is split over two batches B_k and B_{k+1} with fractional jobs $\alpha_1 J_i$ and $\alpha_2 J_i$ ($\alpha_1 + \alpha_2 = 1$), and J_j is split over two batches B_{k+1} and B_{k+2} with fractional jobs $\beta_1 J_j$ and $\beta_2 J_j$ ($\beta_1 + \beta_2 = 1$) in an optimal*



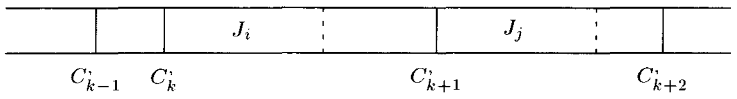
(a) Preemptive batch schedule



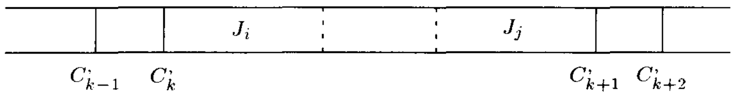
(b) Non-preemptive schedule. Case 1: $\alpha_1 \geq \alpha_2$ and $\beta_1 < \beta_2$



(c) Non-preemptive schedule. Case 2: $\alpha_1 \geq \alpha_2$ and $\beta_1 \geq \beta_2$



(d) Non-preemptive schedule. Case 3: $\alpha_1 < \alpha_2$ and $\beta_1 < \beta_2$



(e) Non-preemptive schedule. Case 4: $\alpha_1 < \alpha_2$ and $\beta_1 \geq \beta_2$

Figure 4.4: Preemptive and non-preemptive schedule, for Lemma 4.5

preemptive schedule. If we move the smaller fractional part of each job J_i and J_j to the batch which contains the other part of the corresponding job, and the change in the total weighted flow time is Δ, then $\Delta < \frac{H_i+H_j}{2}$, where H_i and H_j represent the holding cost of job J_i and J_j respectively in the optimal preemptive schedule.

Proof: Refer to Figure 4.4 (a).

$$\begin{aligned} H_i &\geq \alpha_1 w_i T_k + \alpha_2 w_i (T_k + T_{k+1}) \\ &\geq \alpha_1 w_i (\alpha_1 p_i) + \alpha_2 w_i (\alpha_1 p_i + \alpha_2 p_i + \beta_1 p_j) \\ &= \alpha_1(\alpha_1 + \alpha_2) w_i p_i + \alpha_2^2 w_i p_i + \alpha_2 \beta_1 w_i p_j \\ &= \alpha_1 w_i p_i + \alpha_2^2 w_i p_i + \alpha_2 \beta_1 w_i p_j \\ &> \alpha_1 w_i p_i + \alpha_2 \beta_1 w_i p_j \qquad (4.6) \end{aligned}$$

$$\begin{aligned} H_j &\geq \beta_1 w_j (T_k + T_{k+1}) + \beta_2 w_j (T_k + T_{k+1} + T_{k+2}) \\ &= (\beta_1 + \beta_2)(w_j T_k + w_j T_{k+1}) + \beta_2 w_j T_{k+2} \end{aligned}$$

But applying Lemma 4.3 to B_{k+1} and B_{k+2}, we get $w_j T_{k+2} = p_j W_{k+1}$. Substituting this and using $\alpha_1 + \alpha_2 = 1$ and $\beta_1 + \beta_2 = 1$, we obtain

$$\begin{aligned} H_j &\geq w_j T_k + w_j T_{k+1} + \beta_2 p_j W_{k+1} \\ &\geq w_j(\alpha_1 p_i) + w_j(\alpha_2 p_i + \beta_1 p_j) + \beta_2 p_j (\alpha_2 w_i + \beta_1 w_j) \\ &= \alpha_1 w_j p_i + \alpha_2 w_j p_i + \beta_1 w_j p_j + \beta_2 \alpha_2 w_i p_j + \beta_1 \beta_2 w_j p_j \\ &= w_j p_i + \beta_1 w_j p_j + \beta_2 \alpha_2 w_i p_j + \beta_1 \beta_2 w_j p_j \\ &> \beta_1 w_j p_j + \beta_2 \alpha_2 w_i p_j \qquad (4.7) \end{aligned}$$

Adding inequalities (4.6) and (4.7), we have

$$\begin{aligned} H_i + H_j &> \alpha_1 w_i p_i + \alpha_2 \beta_1 w_i p_j + \beta_1 w_j p_j + \beta_2 \alpha_2 w_i p_j \\ &> \alpha_1 w_i p_i + \alpha_2 w_i p_j + \beta_1 w_j p_j \\ &\geq 2(min\{\alpha_1, \alpha_2\} \alpha_1 w_i p_i + \min\{\beta_1, \beta_2\} \alpha_2 w_i p_j + \qquad (4.8) \\ &\quad min\{\beta_1, \beta_2\} \beta_1 w_j p_j) \qquad (4.9) \end{aligned}$$

The last inequality holds, since $\min\{\alpha_1, \alpha_2\} \leq \frac{1}{2}$ and $\min\{\beta_1, \beta_2\} \leq \frac{1}{2}$. Let the change in the weighted flow time due to moving fractional jobs of J_i and J_j be Δ.

Case 1: $\alpha_1 \geq \alpha_2$; $\beta_1 < \beta_2$

Move $\alpha_2 J_i$ from B_{k+1} to B_k and $\beta_1 J_j$ from B_{k+1} to B_{k+2}. (See Figure 4.4 (b) which shows the new batch completion times C_k' and C_{k+1}'.)

$$\Delta = W_k \alpha_2 p_i - \alpha_2 w_i (T_{k+1} - \alpha_2 p_i) + \beta_1 w_j T_{k+2} - (W_{k+1} - \alpha_2 w_i - \beta_1 w_j)\beta_1 p_j$$

But $W_k p_i = T_{k+1} w_i$ and $W_{k+1} p_j = T_{k+2} w_j$ by Lemma 4.3, so substituting these and using equation (4.9), we get

$$\begin{aligned} \Delta &= \alpha_2^2 w_i p_i + \beta_1^2 w_j p_j + \beta_1 \alpha_2 w_i p_j \qquad (4.10) \\ &< \alpha_1 \alpha_2 w_i p_i + \beta_1^2 w_j p_j + \beta_1 \alpha_2 w_i p_j \\ &< 0.5\alpha_1 w_i p_i + 0.5\beta_1 w_j p_j + 0.5\alpha_2 w_i p_j \\ &< \frac{1}{2}(H_i + H_j). \end{aligned}$$

Case 2: $\alpha_1 \geq \alpha_2$; $\beta_1 \geq \beta_2$

Move $\alpha_2 J_i$ from B_{k+1} to B_k and $\beta_2 J_j$ from B_{k+2} to B_{k+1}. (See Figure 4.4 (c) for the changed completion times.)

$$\Delta = W_k \alpha_2 p_i - \alpha_2 w_i (T_{k+1} - \alpha_2 p_i) + (W_{k+1} - \alpha_2 w_i)\beta_2 p_j - \beta_2 w_j (T_{k+2} - \beta_2 p_j)$$

But $W_k p_i = T_{k+1} w_i$ and $W_{k+1} p_j = T_{k+2} w_j$ by Lemma 4.3, so substituting these and using equation (4.9), we get

$$\begin{aligned} \Delta &= \alpha_2 W_k p_i - \alpha_2 W_k p_i + \alpha_2^2 w_i p_i + \beta_2 W_{k+1} p_j - \alpha_2 \beta_2 w_i p_j - \beta_2 W_{k+1} p_j + \beta_2^2 w_j p_j \\ &= \alpha_2^2 w_i p_i - \alpha_2 \beta_2 w_i p_j + \beta_2^2 w_j p_j \qquad (4.11) \\ &\leq \alpha_1 \alpha_2 w_i p_i + \beta_1 \beta_2 w_j p_j \\ &< 0.5\alpha_1 w_i p_i + 0.5\beta_1 w_j p_j \\ &< \frac{1}{2}(H_i + H_j). \end{aligned}$$

Case 3: $\alpha_1 < \alpha_2$; $\beta_1 < \beta_2$

Move $\alpha_1 J_i$ from B_k to B_{k+1} and $\beta_1 J_j$ from B_{k+1} to B_{k+2}. (See Figure 4.4 (d) for the changed completion times.)

$$\Delta = -(W_k - \alpha_1 w_i)\alpha_1 p_i + \alpha_1 w_i(T_{k+1} - \beta_1 p_j) - (W_{k+1} - \beta_1 w_j)\beta_1 p_j + \beta_1 w_j T_{k+2}$$

But $W_k p_i = T_{k+1} w_i$ and $W_{k+1} p_j = T_{k+2} w_j$ by Lemma 4.3, thus using these and equation (4.9), we get

$$\begin{aligned}\Delta &= -\alpha_1 W_k p_i + \alpha_1^2 w_i p_i + \alpha_1 w_i T_{k+1} - \alpha_1\beta_1 w_i p_j - \beta_1 W_{k+1} p_j + \beta_1^2 w_j p_j + \beta_1 w_j T_{k+2} \\ &= \alpha_1^2 w_i p_i - \alpha_1\beta_1 w_i p_j + \beta_1^2 w_j p_j \qquad (4.12) \\ &\le 0.5\alpha_1 w_i p_i + 0.5\beta_1 w_j p_j \\ &< \frac{1}{2}(H_i + H_j).\end{aligned}$$

Case 4: $\alpha_1 < \alpha_2$; $\beta_1 \ge \beta_2$

Move $\alpha_1 J_i$ from B_k to B_{k+1} and $\beta_2 J_j$ from B_{k+2} to B_{k+1}. (See Figure 4.4 (e) for the changed completion times.)

$$\Delta = -(W_k - \alpha_1 w_i)\alpha_1 p_i + \alpha_1 w_i(T_{k+1} + \beta_2 p_j) + W_{k+1}\beta_2 p_j - \beta_2 w_j(T_{k+2} - \beta_2 p_j)$$

But $W_k p_i = T_{k+1} w_i$ and $W_{k+1} p_j = T_{k+2} w_j$ by Lemma 4.3, thus substituting and using equation (4.9), we obtain

$$\begin{aligned}\Delta &= -\alpha_1 W_k p_i + \alpha_1^2 w_i p_i + \alpha_1 w_i T_{k+1} + \alpha_1\beta_2 w_i p_j + \beta_2 W_{k+1} p_j - \beta_2 w_j T_{k+2} + \beta_2^2 w_j p_j \\ &= \alpha_1^2 w_i p_i + \alpha_1\beta_2 w_i p_j + \beta_2^2 w_j p_j \qquad (4.13) \\ &\le 0.5\alpha_1 w_i p_i + 0.5\alpha_2 w_i p_j + 0.5\beta_1 w_j p_j \\ &< \frac{1}{2}(H_i + H_j).\end{aligned}$$

Therefore, in all possible cases $\Delta < \frac{H_i + H_j}{2}$, as claimed. □

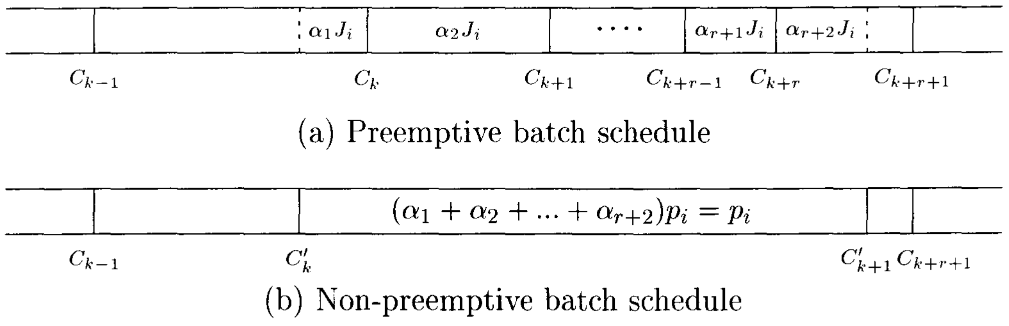


Figure 4.5: Preemptive and non-preemptive schedule for Lemma 4.6

Lemma 4.6 *Consider a job J_i which is scheduled over $r+2$ batches ($r \geq 1$), $B_k, B_{k+1}, \ldots, B_{k+r+1}$ in the optimal preemptive schedule with fractions $\alpha_1, \alpha_2, \ldots, \alpha_{r+2}$ (where $\alpha_1 + \alpha_2 + \cdots + \alpha_{r+2} = 1$). If we combine all these fractional parts of job J_i into batch B_{k+1}, then the change in the total weighted flow time is less than H_i.*

Proof: Refer to Figure 4.5.

$$\begin{aligned} H_i &\geq \alpha_1 w_i(T_k + \alpha_1 p_i) + \alpha_2 w_i(T_k + \alpha_1 p_i + \alpha_2 p_i) + \ldots \\ &\quad + \alpha_{r+2} w_i(T_k + \alpha_1 p_i + \alpha_2 p_i + \cdots + \alpha_{r+2} p_i) \\ &\geq \alpha_1 w_i(\alpha_1 p_i) + \alpha_2 w_i(\alpha_1 p_i + \alpha_2 p_i) + \cdots + \alpha_{r+2} w_i(\alpha_1 p_i + \alpha_2 p_i + \cdots + \alpha_{r+2} p_i) \end{aligned}$$

Let the change in the weighted flow time due to combining the fractional jobs of job J_i into batch B_{k+1} be Δ.

When we move $\alpha_1 J_i$, $\alpha_2 J_i, \ldots, \alpha_{r+2} J_i$ to batch B_{k+1}, only the flow times of batches B_k, $B_{k+1}, \ldots, B_{k+r+1}$ will be changed. Flow times of other batches will remain the same. In the new schedule, batch B_k will have total weight of $(W_k - \alpha_1 w_i)$ and it will be completed at time $C'_k = (C_k - \alpha_1 p_i)$; batch B_{k+1} will have total weight of $(\alpha_1 w_i + \alpha_2 w_i + \cdots + \alpha_{r+2} w_i) = w_i$ and it will be completed at time $C'_{k+1} = C_k + (\alpha_2 p_i + \cdots + \alpha_{r+2} p_i)$; if $r \geq 2$, batches $B_{k+2}, B_{k+3}, \ldots, B_{k+r}$ will be empty; batch B_{k+r+1} will have total weight of $(W_{k+r+1} - \alpha_{r+2} w_i)$ and will be completed at C_{k+r+1}.

Therefore,

$$\begin{aligned}\Delta &= \alpha_1 w_i[\alpha_2 p_i + \alpha_3 p_i + \cdots + \alpha_{r+2} p_i] + \alpha_2 w_i[\alpha_3 p_i + \alpha_4 p_i + \cdots + \alpha_{r+2} p_i] \\ &\quad + \cdots + \alpha_{r+1} w_i[\alpha_{r+2} p_i] - (W_k - \alpha_1 w_i)\alpha_1 p_i - \alpha_{r+2} w_i(T_{k+r+1} - \alpha_{r+2} p_i)\end{aligned}$$

In the above equation, the first term is the increase in the weighted flow time of partial job $\alpha_1 J_i$ due to the increase in its completion time by $(\alpha_2 p_i + \alpha_3 p_i + \cdots + \alpha_{r+2} p_i)$; the second term is the increase in the weighted flow time of the partial job $\alpha_2 J_i$ due to the increase in its completion time by $(\alpha_3 p_i + \cdots + \alpha_{r+2} p_i)$; the third term is the increase in the weighted flow time of the partial job $\alpha_{r+1} J_i$ due to the increase in its completion time by $\alpha_{r+2} p_i$; the fourth term is the decrease in the weighted flow time of jobs in the original batch B_k, except the partial job $\alpha_1 J_i$, due to the decrease in its completion time by $\alpha_1 p_i$; the fifth term is the decrease in the weighted flow time of partial job $\alpha_{r+2} J_i$ due to the decrease in its completion time by $(T_{k+r+1} - \alpha_{r+2} p_i)$.

$$\begin{aligned}\Delta &= \alpha_1 w_i[\alpha_2 p_i + \alpha_3 p_i + \cdots + \alpha_{r+2} p_i] + \alpha_2 w_i[\alpha_3 p_i + \alpha_4 p_i + \cdots + \alpha_{r+2} p_i] \\ &\quad + \cdots + \alpha_{r+1} w_i[\alpha_{r+2} p_i] - (W_k - \alpha_1 w_i)\alpha_1 p_i - \alpha_{r+2} w_i(T_{k+r+1} - \alpha_{r+2} p_i) \\ &= \alpha_2 w_i(\alpha_1 p_i) + \alpha_3 w_i(\alpha_1 p_i + \alpha_2 p_i) + \cdots + \alpha_{r+2} w_i(\alpha_1 p_i + \alpha_2 p_i + \cdots + \alpha_{r+1} p_i) \\ &\quad -(W_k - \alpha_1 w_i)\alpha_1 p_i - \alpha_{r+2} w_i(T_{k+r+1} - \alpha_{r+2} p_i)\end{aligned}$$

But $W_k \geq \alpha_1 w_i$ and $T_{k+r+1} \geq \alpha_{r+2} p_i$, therefore

$$\begin{aligned}\Delta &\leq \alpha_2 w_i(\alpha_1 p_i) + \alpha_3 w_i(\alpha_1 p_i + \alpha_2 p_i) + \cdots + \alpha_{r+2} w_i(\alpha_1 p_i + \alpha_2 p_i + \cdots + \alpha_{r+1} p_i) \\ &< H_i \quad \square\end{aligned}$$

Lemma 4.7 *Any algorithm which rounds preempted jobs of the optimal preemptive batch schedule according to Lemmas 4.4 - 4.6 is a 2-approximation algorithm.*

Proof: From Lemmas 4.4, 4.5 and 4.6, the total change in the weighted flow time due to these roundings of jobs is less than $\sum_i^N H_i$. But the total weighted flow time of the optimal preemptive schedule is $TC_{pmt} = \sum_{i=1}^{N} H_i$. Therefore, the total weighted flow time after rounding the preempted jobs into non-preempted jobs is less than $2TC_{pmt}$. Also note that the total delivery cost after rounding is not more than nd (which is the delivery cost of the optimal preemptive batch schedule), where n is the total number of batches in the optimal preemptive schedule. Thus the total cost of the rounded schedule is less than $2TC^*$, where $TC^* \geq TC_{pmt} + nd$ is the total cost of the optimal *non-preemptive* schedule. □

The following result shows that we do *not* need to know the optimal preemptive schedule, it is used only for proving the upper bound for our much simpler approximation.

Corollary 4.1 *Optimal batching of the WSPT job sequence is a 2-approximation algorithm.*

Proof: It is obvious that the rounding of the preempted jobs as explained in Lemmas 4.4 - 4.6 does not disturb the job sequence. Furthermore, in the optimal preemptive batch schedule, jobs follow the WSPT job sequence. Therefore, the non-preemptive batch schedule of Lemma 4.7 follows the WSPT job sequence. Thus, the optimal non-preemptive batching of the WSPT job sequence will not be worse than this rounded solution. By Lemma 4.7, the rounded batch schedule is a 2-approximation solution, and so the optimal batching of the WSPT sequence cannot be worse. □

4.2.3 Approximation Algorithm

Corollary 4.1 allows us to use the following simple approximation algorithm.

Algorithm 4.1 (Approximation algorithm)

```
Step 1   Sequence the jobs in WSPT order
Step 2:  Call Algorithm 3.2 for the optimal batching of this job sequence.
```

Theorem 4.1 *Algorithm 4.1 is a 2-approximation algorithm with time complexity $O(NlogN)$.*

Proof: We know from Corollary 4.1 that Algorithm 4.1 is a 2-approximation algorithm. As discussed in Section 3.4.1, the optimal batching of the given WSPT job sequence can be obtained in $O(N)$ time, and obtaining the WSPT job sequence requires $O(NlogN)$ time. □

Remark 4.2 In the preemptive optimal batch schedule, if jobs are preempted at most into two batches, then Algorithm 4.1 is a $\frac{3}{2}$-approximation algorithm (from Lemmas 4.4, 4.5).

4.2.4 Delivery Cost and Performance of Algorithm 4.1

In this section, we study the performance of Algorithm 4.1 with different values of the delivery cost d. We show that the batch schedule generated by the algorithm is close to the optimal batch schedule when the batch delivery cost is relatively large. Generally, the delivery cost is larger than the holding cost of an item. For example, the batch delivery cost can be the combination of the transportation cost, loading and unloading cost; but the weight w_i of any job J_i is the holding cost of that job over a unit time. Thus, we often have $(wp)_{max} \leq d$, where $(wp)_{max} = \max_{i,j=1,2,\ldots,N} \{w_i p_j\}$. We further analyze below the performance of Algorithm 4.1 when $(wp)_{max} \leq d$. Let $R = \frac{d}{(wp)_{max}}$, where $R \geq 1$ by assumption.

Lemma 4.8 *In the optimal preemptive batching schedule, a job J_i is split into at most two batches if $R \geq 1$.*

Proof: Let us assume that there exists an optimal schedule in which a job J_i is split into $(r+2)$ batches, $B_k, B_{k+1}, \ldots, B_{k+r+1}$, $r \geq 1$ (refer to Figure 4.5 (a)). Move partial jobs $\alpha_2 J_i$, $\alpha_3 J_i, \ldots, \alpha_{r+1} J_i$ to batch B_k. Let the change in the holding cost due to this move be Δ.

$$\begin{aligned}\Delta &= W_k(\alpha_2 p_i + \alpha_3 p_i + \cdots + \alpha_{r+1} p_i) + \alpha_2 w_i(\alpha_3 p_i + \alpha_4 p_i + \cdots + \alpha_{r+1} p_i) \\ &\quad + \alpha_3 w_i(\alpha_4 p_i + \alpha_5 p_i + \cdots + \alpha_{r+1} p_i) + \cdots + \alpha_r w_i \alpha_{r+1} p_i - rd\end{aligned}$$

From Lemma 4.3, $\alpha_{j+1} p_i = T_{k+j} = \frac{p_i W_{k+j-1}}{w_i} = \frac{\alpha_j w_i p_i}{w_i} = \alpha_j p_i$, for $j = 2, 3, ..., r$, which implies $\alpha_2 = \alpha_3 = \cdots = \alpha_{r+1}$. From $W_k p_i = T_{k+1} w_i = \alpha_2 p_i w_i$, we also have $W_k = \alpha_2 w_i$. Let $\alpha_2 = \alpha_3 = \cdots = \alpha_{r+1} = \alpha$, then

$$\begin{aligned}\Delta &= \alpha^2 w_i p_i((r) + (r-1) + \cdots + 2 + 1) - rd \\ &= \alpha^2 w_i p_i \frac{r}{2}(r+1) - rd \\ &\leq \alpha^2 (wp)_{max} \frac{r}{2}(r+1) - rd \\ \text{But } \alpha &\leq \frac{1}{r}, \\ \text{therefore, } \Delta &\leq (wp)_{max} \frac{r+1}{2r} - rd \\ &\leq \frac{r+1}{2r} d - rd < 0.\end{aligned}$$

This contradicts the optimality assumption. □

Lemma 4.9 *If $R \geq 1$, then total change in the holding cost from rounding the preempted jobs in the optimal preemptive schedule is less than $\frac{3(n-1)}{8}(wp)_{max}$, where n is the number of batches in the optimal preemptive schedule.*

Proof: From Lemma 4.8, we know a job is split into at most two batches. Therefore preempted jobs are rounded using Lemmas 4.4 and 4.5. In Lemma 4.4, only one job J_i $(i = 1, 2, ..., N)$ is rounded and the change in the weighted flow time is less than

$\frac{1}{4}w_ip_i \le \frac{1}{4}(wp)_{max}$ (refer equations (4.4) and (4.5)). In Lemma 4.5, two preempted jobs J_i and J_j are rounded and the change in the weighted flow time per job rounded is less than $\frac{1}{8}(w_ip_i + w_jp_j + w_ip_j) \le \frac{3}{8}(wp)_{max}$ (refer to equations (4.10) - (4.13)). Therefore, change in the weighted flow time per job rounded is less than $\frac{3}{8}(wp)_{max}$. Since there are n batches, there will be at most $(n-1)$ roundings. Thus, total change in the holding cost is less than $\frac{3(n-1)}{8}(wp)_{max}$. □

Theorem 4.2 *For a given $R \ge 1$, Algorithm 4.1 is a $\frac{9+8R}{6+8R}$-approximation algorithm.*

Proof: From Lemma 4.8, any job is split into at most two batches in the optimal preemptive schedule. Let Δ denote the total change in the weighted flow time from rounding these split jobs and let ρ be the approximation ratio of Algorithm 4.1. From Lemma 4.9, $\Delta \le \frac{3(n-1)}{8}(wp)_{max}$ and from Remark 2, $\Delta < \frac{1}{2}TC_{pmt}$. Since the gap between the objective value produced by Algorithm 4.1 and TC^* will be no more than Δ, we have $\rho \le \frac{TC^*+\Delta}{TC^*}$. Furthermore,

$$\begin{aligned}
\Delta &< \frac{3(n-1)}{8}(wp)_{max} \le \frac{3(n-1)d}{8R} \\
\text{Therefore, } (n-1)d &> \frac{8R\Delta}{3} \\
\text{Also } TC_{pmt} &> 2\Delta \\
TC^* &\ge TC_{pmt} + nd \\
TC^* &> 2\Delta + nd \\
&> 2\Delta + (n-1)d \\
&> 2\Delta + \frac{8R\Delta}{3} \\
&> \frac{(6+8R)\Delta}{3}
\end{aligned}$$

This implies $\frac{\Delta}{TC^*} < \frac{3}{6+8R}$ and $\rho \le 1 + \frac{\Delta}{TC^*} < \frac{9+8R}{6+8R}$. □

Table 4.1 shows how the approximation guarantee changes as the R value increases.

$R = \frac{d}{(wp)_{max}}$	ρ
1.00	1.214
2.00	1.136
3.00	1.100
4.00	1.079
5.00	1.065

Table 4.1: Performance of the algorithm with different $R \geq 1$ values.

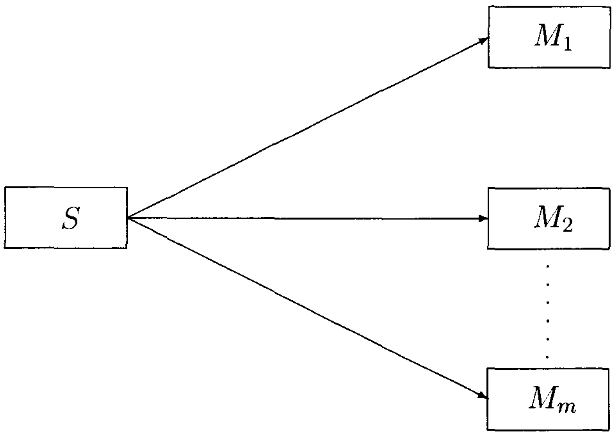


Figure 4.6: Multiple Customer Model.

4.3 Multiple Products, Multiple Customers

In this section, we study batch scheduling at the supplier who supplies products to m customers $M_1, M_2, \ldots, M_m$ (refer to Figure 4.6). In fact, these customers may be manufacturers in the supply chain. A batch schedule with multiple customers will have the job sequence σ of all the jobs, job sequence $\sigma^{(k)}$ of customer M_k, $k = 1, 2, \ldots, m$, and the batch sequence. Since we deal with multiple customers, we need an additional index to denote the customer.

In section 3.3, we gave an $O(\sum_{k=1}^{m} \sum_{i=1}^{S} log(N_k))$ algorithm for multiple customer batch scheduling problem of identical product for each customer M_k, where N_k is the number of items to be delivered to customer M_k, and S is the number of product types. Hall and Potts [19] provide a dynamic programming algorithm with time complexity $O((\sum_{k=1}^{m} N_k)^{m+1})$ to solve multi-customer batch scheduling problems with non-identical products to minimize the sum of the completion time and total delivery cost. The multiple customer batch scheduling problem with arbitrary processing times and arbitrary weights is NP-hard. We prove that if each $\sigma^{(k)}$ $(k = 1, 2, \ldots, m)$ follows the WSPT job sequence *(WSPT customer job sequence)*, then the optimal batching on these job sequences is a 2-approximation solution. We first analyze the preemptive version of the problem and then we round the preempted jobs to get the 2-approximation algorithm.

4.3.1 Preemptive Batch Scheduling of Multiple Products for Multiple Customers

In this section, we study some properties of preemptive batch schedules of the multiple customer problem.

Property 4.1 *Changing the sequence of jobs within a given batch will not affect the flow time of any jobs in the schedule.*

Therefore, there exists an optimal preemptive batch schedule in which jobs in the same batch are scheduled in WSPT order. Hereafter, whenever we talk about a preemptive batch schedule, it is assumed that jobs within the same batch are scheduled in WSPT order.

Lemma 4.10 *There exists an optimal preemptive batch schedule in which jobs of the same customer are scheduled in WSPT order.*

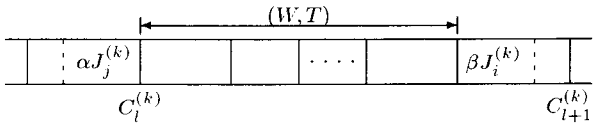


Figure 4.7: Partial schedule for Lemma 4.10.

Proof: If the lemma is not true then there will be at least two jobs $\beta J_i^{(k)}$ and $\alpha J_j^{(k)}$, $(0 < \alpha, \beta \leq 1)$ for some $k = 1, 2, \ldots, m$ such that $\beta J_i^{(k)}$ immediately follows $\alpha J_j^{(k)}$ in $\sigma^{(k)}$ and $\frac{p_j^{(k)}}{w_j^{(k)}} > \frac{p_i^{(k)}}{w_i^{(k)}}$. Thus, $\beta J_i^{(k)}$ and $\alpha J_j^{(k)}$ may be assigned to a batch $B_l^{(k)}$, or assigned to batches $B_l^{(k)}$ and $B_{l+1}^{(k)}$ respectively.

If they both are assigned to $B_l^{(k)}$, then from Property 4.1, interchanging $\alpha J_j^{(k)}$ and $\beta J_i^{(k)}$ will not affect the total cost. If $\beta J_i^{(k)}$ and $\alpha J_j^{(k)}$ are assigned to $B_l^{(k)}$ and $B_{l+1}^{(k)}$, respectively, refer to Figure 4.7, where W and T denote the total weight and total processing time, respectively, of all the batches which are scheduled between $B_l^{(k)}$ and $B_{l+1}^{(k)}$.

Case 1: $\alpha p_j^{(k)} \geq \beta p_i^{(k)}$

Move $\beta J_i^{(k)}$ to $B_l^{(k)}$ and $\frac{\beta p_i^{(k)}}{p_j^{(k)}}$ of $\alpha J_j^{(k)}$ to $B_{l+1}^{(k)}$. This will not change the batch completion times. However, $W_l^{(k)}$ is increased and $W_{l+1}^{(k)}$ is decreased by $\beta w_i^{(k)} - \frac{\beta p_i^{(k)}}{p_j^{(k)}} w_j^{(k)}$. Thus the interchange has decreased the holding cost, a contradiction.

Case 2: $\alpha p_j^{(k)} < \beta p_i^{(k)}$

Move $\frac{\alpha p_j^{(k)}}{p_i^{(k)}} J_i^{(k)}$ of $\beta J_i^{(k)}$ to $B_l^{(k)}$ and $\alpha J_j^{(k)}$ to $B_{l+1}^{(k)}$. Again this will not affect the batch completion time, but $W_l^{(k)}$ is increased and $W_{l+1}^{(k)}$ is decreased by $\frac{\alpha p_j^{(k)}}{p_i^{(k)}} w_i^{(k)} - \alpha w_j^{(k)}$. Thus the interchange has decreased the holding cost.

In both cases, the number of batches is not changed but the holding cost is decreased. This contradicts the optimality assumption. □

Remark 4.3 There exists an optimal preemptive batch schedule, in which if a job $J_i^{(k)}$ is split into two batches $B_r^{(k)}$ and $B_s^{(k)}$, then $s = r + 1$, i.e., no batch of customer M_k is scheduled between $B_r^{(k)}$ and $B_s^{(k)}$. Thus, in the optimal preemptive batch schedule,

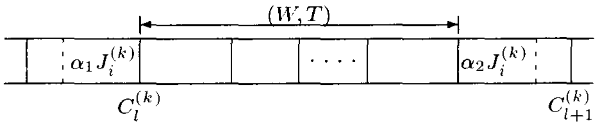


Figure 4.8: Partial schedule when $\alpha_1 J_i^{(k)}$ and $\alpha_2 J_i^{(k)}$ are assigned to $B_l^{(k)}$ and $B_{l+1}^{(k)}$.

job $J_i^{(k)}$ $(k = 1, 2, \ldots, m)$ may be assigned to a single batch $B_l^{(k)}$; assigned to two batches $B_l^{(k)}$ and $B_{l+1}^{(k)}$; or assigned to $r+2$ batches $(r \geq 1)$, $B_l^{(k)}, B_{l+1}^{(k)}, \ldots, B_{l+r+1}^{(k)}$, with no other job for the same customer scheduled between them.

Lemma 4.11 *If a job $J_i^{(k)}$ is split into two batches, $B_l^{(k)}$ and $B_{l+1}^{(k)}$ in the optimal preemptive schedule, then $(W_l^{(k)} + W)p_i^{(k)} = w_i^{(k)}(T + T_{l+1}^{(k)})$, where W and T are respectively the total weight and total processing time of all the batches which are scheduled between batches $B_l^{(k)}$ and $B_{l+1}^{(k)}$.*

Proof: Refer to Figure 4.8. Let us assume that job $J_i^{(k)}$ is split into two batches $B_l^{(k)}$ and $B_{l+1}^{(k)}$ with $\alpha_1 J_i^{(k)}$ and $\alpha_2 J_i^{(k)}$ respectively, where $\alpha_1 + \alpha_2 \leq 1$.

Case 1: Move $\delta J_i^{(k)}$ $(0 < \delta \leq \alpha_1)$ from $B_l^{(k)}$ to $B_{l+1}^{(k)}$. Let the change in the weighted flow time be Δ_1.

$$\begin{aligned} \Delta_1 &= \delta w_i^{(k)}(T + T_{l+1}^{(k)}) - (W_l^{(k)} - \delta w_i^{(k)} + W)\delta p_i^{(k)} \\ \text{But } \Delta_1 &\geq 0 \text{ by optimality.} \\ \text{Therefore, } w_i^{(k)}(T + T_{l+1}^{(k)}) &\geq (W_l^{(k)} - \delta w_i^{(k)} + W)p_i^{(k)} \end{aligned}$$

Since the last inequality holds for arbitrarily small $\delta > 0$, we must have

$$w_i^{(k)}(T + T_{l+1}^{(k)}) \geq (W_l^{(k)} + W)p_i^{(k)} \tag{4.14}$$

Case 2: Move $\delta J_i^{(k)}$ $(0 < \delta \le \alpha_2)$ from $B_{l+1}^{(k)}$ to $B_l^{(k)}$ and let the change in the weighted flow time be Δ_2.

$$\begin{aligned}\Delta_2 &= -\delta w_i^{(k)}(T + T_{l+1}^{(k)} - \delta p_i^{(k)}) + (W_l^{(k)} + W)\delta p_i^{(k)} \\ \text{But } \Delta_2 &\ge 0 \text{ by optimality.} \\ \text{Therefore, } w_i^{(k)}(T + T_{l+1}^{(k)} - \delta p_i^{(k)}) &\le (W_l^{(k)} + W)p_i^{(k)}\end{aligned}$$

Since the last inequality holds for arbitrary small $\delta > 0$, we must have

$$w_i^{(k)}(T + T_{l+1}^{(k)}) \le (W_l^{(k)} + W)p_i^{(k)} \tag{4.15}$$

From inequalities (4.14) and (4.15), $(W_l^{(k)} + W)p_i^{(k)} = w_i^{(k)}(T + T_{l+1}^{(k)})$. □

In the next section, we develop the 2-approximation algorithm using the properties the optimal preemptive batch schedule discussed in this section.

4.3.2 Nonpreemptive Batch Scheduling of Multiple Products for Multiple Customers

We prove the existence of 2-approximation algorithm for this problem.

Lemma 4.12 *Consider a job $J_i^{(k)}$ of customer M_k split over two batches $B_l^{(k)}$ and $B_{l+1}^{(k)}$ with fractional jobs $\alpha_1 J_i^{(k)}$ and $\alpha_2 J_i^{(k)}$. Let Δ be the change in the weighted flow time when moving $\alpha_1 J_i^{(k)}$ to $B_{l+1}^{(k)}$. Then $\Delta < w_i^{(k)} p_i^{(k)}$, and $\Delta < H_i^{(k)}$, where $H_i^{(k)}$ is the holding cost of $J_i^{(k)}$ in the optimal preemptive schedule.*

Proof: Refer to Figure 4.8.

$$\Delta \le -(W + W_l^{(k)} - \alpha_1 w_i^{(k)})\alpha_1 p_i^{(k)} + \alpha_1 w_i^{(k)}(T + T_{l+1}^{(k)})$$

From Lemma 4.11, $(W_l^{(k)} + W)p_i^{(k)} = w_i^{(k)}(T + T_{l+1}^{(k)})$. Therefore,

$\Delta \le \alpha_1^2 w_i^{(k)} p_i^{(k)}$

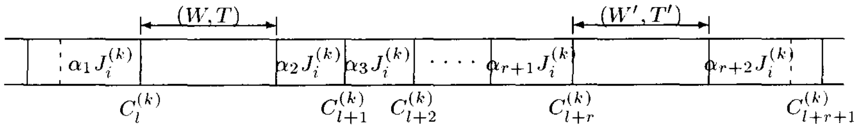


Figure 4.9: Partial schedule when $J_i^{(k)}$ is split into $r+2$ batches $(r \geq 1)$.

But $H_i^{(k)} \geq \alpha_1 w_i^{(k)}(\alpha_1 p_i^{(k)}) + \alpha_2 w_i^{(k)}(\alpha_1 p_i^{(k)} + \alpha_2 p_i^{(k)})$

Therefore, $\Delta < H_i^{(k)}$.

Since $\alpha_1 < 1$, $\Delta < w_i^{(k)} p_i^{(k)}$ follows too. □

As in WSPT job order, we also define *batch-WSPT order* where batches are sequenced in increasing order of $\frac{T_i}{W_i}$ ratio.

Property 4.2 *In the optimal preemptive batching schedule, batches are scheduled in batch-WSPT order.*

Lemma 4.13 *If a job $J_i^{(k)}$ is split into $r+2$ batches $(r \geq 1)$ with $\alpha_1 J_i^{(k)}, \alpha_2 J_i^{(k)}, \ldots, \alpha_{r+2} J_i^{(k)}$, then there exists an optimal preemptive batch schedule in which, $\alpha_2 J_i^{(k)}, \alpha_3 J_i^{(k)}, \ldots, \alpha_{r+1} J_i^{(k)}$ are scheduled in consecutive batches.*

Proof: Refer to Figure 4.9. Note that batches $B_{l+1}^{(k)}, B_{l+2}^{(k)}, \ldots, B_{l+r}^{(k)}$ will have the same ratio $\frac{p_i^{(k)}}{w_i^{(k)}}$ by Property 4.2. If there is any batch $B_s^{(u)}$ of customer M_u that is assigned between two of these batches, then $B_s^{(u)}$ too will have the same ratio $\frac{p_i^{(k)}}{w_i^{(k)}}$. Therefore, moving this batch $B_s^{(u)}$ after batch $B_{l+r}^{(k)}$ will not change the holding cost.

□

Lemma 4.14 *In the optimal preemptive batch schedule, consider any job $J_i^{(k)}$ which is split into $r+2$ batches $(r \geq 1)$ $B_l^{(k)}, B_{l+1}^{(k)}, \ldots, B_{l+r+1}^{(k)}$. If we schedule the whole job $J_i^{(k)}$ into batch $B_{l+1}^{(k)}$, then Δ, the total change in the holding cost due to this adjustment, is less than $H_i^{(k)}$ and $\Delta < w_i^{(k)} p_i^{(k)}$.*

Proof: Refer to Figure 4.9. Let the change in the total holding cost due to moving all the fractional jobs of $J_i^{(k)}$ to $B_{l+1}^{(k)}$ be Δ.

$$\begin{aligned}
\Delta &= \alpha_1 w_i^{(k)}[T - \alpha_1 p_i^{(k)} + (\alpha_2 + \alpha_3 + \cdots + \alpha_{r+2})p_i^{(k)}] + \alpha_2 w_i^{(k)}(\alpha_3 + \alpha_4 + \cdots + \alpha_{r+2})p_i^{(k)} \\
&\quad + \alpha_3 w_i^{(k)}(\alpha_4 + \alpha_5 + \cdots + \alpha_{r+2})p_i^{(k)} + \cdots + \alpha_{r+1} w_i^{(k)} \alpha_{r+2} p_i^{(k)} \\
&\quad - (W + W_l^{(k)} - \alpha_1 w_i^{(k)})\alpha_1 p_i^{(k)} - \alpha_{r+2} w_i^{(k)}(T' + T_{l+r+1}^{(k)} - \alpha_{r+2} p_i^{(k)}) + W' \alpha_{r+2} p_i^{(k)} \\
\Delta &= \alpha_1 w_i^{(k)}(T + \alpha_2 p_i^{(k)}) - \alpha_1^2 w_i^{(k)} p_i^{(k)} + \alpha_1 w_i^{(k)}(\alpha_3 + \cdots + \alpha_{r+2})p_i^{(k)} \\
&\quad + \alpha_2 w_i^{(k)}(\alpha_3 + \alpha_4 + \cdots + \alpha_{r+2})p_i^{(k)} + \alpha_3 w_i^{(k)}(\alpha_4 + \alpha_5 + \cdots + \alpha_{r+2})p_i^{(k)} \\
&\quad + \cdots + \alpha_{r+1} w_i^{(k)} \alpha_{r+2} p_i^{(k)} - (W + W_l^{(k)})\alpha_1 p_i^{(k)} + \alpha_1^2 w_i^{(k)} p_i^{(k)} \\
&\quad - \alpha_{r+2} w_i^{(k)}(T' + T_{l+r+1}^{(k)}) + \alpha_{r+2}^2 w_i^{(k)} p_i^{(k)} + W' \alpha_{r+2} p_i^{(k)}
\end{aligned}$$

But from Lemma 4.11,

$$(W_l^{(k)} + W)p_i^{(k)} = w_i^{(k)}(T + \alpha_2 p_i^{(k)}) \text{ and } (\alpha_{r+1} w_i^{(k)} + W')p_i^{(k)} = w_i^{(k)}(T' + T_{l+r+1}^{(k)}).$$

Therefore,

$$\begin{aligned}
\Delta &= \alpha_1 w_i^{(k)}(\alpha_3 + \cdots + \alpha_{r+2})p_i^{(k)} + \alpha_2 w_i^{(k)}(\alpha_3 + \alpha_4 + \cdots + \alpha_{r+2})p_i^{(k)} \\
&\quad + \alpha_3 w_i^{(k)}(\alpha_4 + \alpha_5 + \cdots + \alpha_{r+2})p_i^{(k)} + \cdots + \alpha_r w_i^{(k)}(\alpha_{r+1} + \alpha_{r+2})p_i^{(k)} + \alpha_{r+2}^2 w_i^{(k)} p_i^{(k)} \\
&= \alpha_3 w_i^{(k)}(\alpha_1 + \alpha_2)p_i^{(k)} + \alpha_4 w_i^{(k)}(\alpha_1 + \alpha_2 + \alpha_3)p_i^{(k)} + \ldots \\
&\quad + \alpha_{r+2} w_i^{(k)}(\alpha_1 + \alpha_2 + \cdots + \alpha_{r+1})p_i^{(k)} \\
&\quad - \alpha_{r+2}\alpha_{r+1} w_i^{(k)} p_i^{(k)} + \alpha_{r+2}^2 w_i^{(k)} p_i^{(k)} \\
&< H_i^{(k)}
\end{aligned}$$

$$\begin{aligned}\Delta &= \alpha_3 w_i^{(k)}(\alpha_1+\alpha_2)p_i^{(k)} + \alpha_4 w_i^{(k)}(\alpha_1+\alpha_2+\alpha_3)p_i^{(k)} + \dots \\ &\quad + \alpha_{r+2} w_i^{(k)}(\alpha_1+\alpha_2+\cdots+\alpha_{r+1})p_i^{(k)} \\ &\quad - \alpha_{r+2}\alpha_{r+1} w_i^{(k)} p_i^{(k)} + \alpha_{r+2}^2 w_i^{(k)} p_i^{(k)} \\ &< (\alpha_1+\alpha_2+\cdots+\alpha_{r+2})^2 w_i^{(k)} p_i^{(k)} \\ &= w_i^{(k)} p_i^{(k)}. \quad \square\end{aligned}$$

Corollary 4.2 *In the preemptive optimal schedule, if any job split into two batches $B_l^{(k)}$ and $B_{l+1}^{(k)}$ is scheduled according to Lemma 4.12 and if any job split into $r+2$ batches $(r \geq 1)$ $B_l^{(k)}, B_{l+1}^{(k)}, \dots, B_{l+r+1}^{(k)}$ is scheduled according to Lemma 4.14, then we will get a non-preemptive batch schedule with 2-approximation.*

Proof: From Lemmas 4.12 and 4.14, total change in the weighted flow time is less than $\sum_{k=1}^{m}\sum_{i=1}^{N_k} H_i^{(k)}$. $\square$

Theorem 4.3 *Any algorithm which can optimally solve the multiple customer batch scheduling problem with fixed WSPT customer job sequence is a 2-approximation algorithm.*

Proof: We know that starting from the preemptive solution on the WSPT customer job sequence, we can get a 2-approximation solution. This 2-approximation solution maintains WSPT job sequence among jobs for the same customer. Therefore, any optimal batching solution on WSPT customer job sequence will not be worse than 2-approximation. $\square$

The dynamic programming algorithm provided by Hall and Potts can be modified for batching a given job sequence with arbitrary processing times and arbitrary weights. This algorithm will take $O((\sum_{k=1}^{m} N_k)^{m+1})$ time.

Theorem 4.4 *Applying the dynamic programming algorithm by Hall and Potts to the WSPT job sequence will provide a 2-approximation solution for the multi-customer batch scheduling problem with arbitrary processing times and arbitrary weights.*

Remark 4.1 If for each customer M_k, $(wp)_{max} \leq d$, then the algorithm performance is similar to the one shown in Table 4.1.

Chapter 5

Batch Scheduling at the Manufacturer

5.1 Introduction

In this chapter, we study the batch scheduling problem at the manufacturer where production is controlled by a push system. There are h end-customers $M^{(m+1)}, M^{(m+2)}, \ldots, M^{(m+h)}$ with $N^{(j)}$ jobs for customer $M^{(j)}$ $(j = m+1, m+2, \ldots, m+h)$. We call these end-customers simply customers in this chapter, and in Chapters 6 and 7. The manufacturer is required to process jobs on a single machine and the kth job of customer $M^{(j)}$, denoted by $J_k^{(j)}$, needs $p_k^{(j)}$ time to process on the machine $(j = m+1, m+2, \ldots, m+h)$. We assume that items (jobs) arrive at the manufacturer at different time points. The item arrival time depends on its immediate upstream supplier's delivery time.

Associated with each batch delivery to customer $M^{(j)}$ is the delivery cost $d^{(j)}$ which is charged to the manufacturer. There incurs a holding cost of $w_k^{(j)}$ per unit time for job $J_k^{(j)}$. Let the number of delivery batches to customer $M^{(j)}$ be $n^{(j)}$ and the ith batch delivered to customer $M^{(j)}$ be $B_i^{(j)}$. We have to find the batch schedule so that the total cost of inventory holding and batch delivery at the manufacturer is minimized, i.e., our objective is to minimize $TC = \sum_{j=m+1}^{m+h} \sum_{l=1}^{n^{(j)}} W_l^{(j)} T_l^{(j)} + \sum_{j=m+1}^{m+h} n^{(j)} d^{(j)}$, where $W_i^{(j)}$ is the sum of weights of all the jobs assigned to batch $B_i^{(j)}$ and $T_i^{(j)}$ is the delivery time of batch $B_i^{(j)}$.

We first focus on the single customer problems and then extend its results to multicustomer problems. Therefore, we remove the superscript (j) representing customer $M^{(j)}$ in our modeling work.

In the manufacturer's problem, jobs arrive in batches at different time points. Therefore, unlike in the supplier's problem, the manufacturer's problem has another constraint that job J_i has release time r_i, for $i = 1, 2, \ldots, N$. First we provide a polynomial algorithm for batch scheduling of a fixed job sequence and single product problems. Batch scheduling at the manufacturer to minimize weighted sum of completion time is NP-hard [19]. It is a hard problem even if the weights $w_i = 1$ $(i = 1, 2, \ldots, N)$.

Therefore, we develop a 2-approximation algorithm for batch scheduling of jobs with identical weights and a hybrid meta-heuristic algorithm for general batch scheduling of multiple products. Our meta-heuristic uses a genetic algorithm. At the end of the chapter, we compare the performance of the hybrid algorithm with an algorithm which provides a lower bound to the problem.

Before developing models for different problems, in the next section, we first analyze some preliminaries of the manufacturer's problem.

5.2 Some Preliminaries for the Manufacturer's Problem

Since jobs have release times, manufacturer's problems are harder than supplier's problems. In this section, we provide some basic properties of the manufacurer's

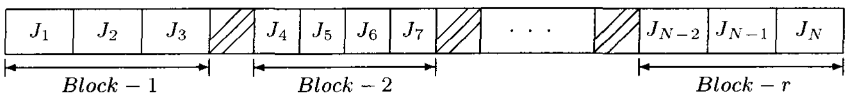


Figure 5.1: A Schedule with r blocks.

problem.

Property 5.1 *There exists an optimal schedule in which a batch is delivered as soon as the last job of the batch has completed processing.*

We call any schedule in which at any time point t when a job starts processing, the first job available from the job list is scheduled for processing a *list schedule.* Without loss of generality, we may assume that the list is $\{J_1, J_2, \ldots, J_N\}$. In a list schedule, there may be an idle time on the machine before starting processing of J_i $(i = 2, 3, \ldots, N)$ if J_{i-1} is completed before J_i is released. We call these idle times inserted on the machine *forced* idle times. A set of jobs assigned between two adjacent forced idle times is called a *block.* For example, in Figure 5.1, there are r blocks in the schedule. $Block - 1 = \{J_1, J_2, J_3\}$, $Block - 2 = \{J_4, J_5, J_6, J_7\}$, and $Block - r = \{J_{N-2}, J_{N-1}, J_N\}$.

We further define a schedule as a *busy schedule*, if the machine is continuously processing jobs from time $t = 0$ until the completion of the last job.

5.3 Batching of Jobs with Fixed Job Sequence

We prove that batch scheduling problems with release times can be solved polynomially when the jobs follow an arbitrary but fixed job sequence. Let us assume, without loss of generality, that the job sequence is $J_1, J_2, \ldots, J_N$. We use the following modification to solve batching of a fixed job sequence. Processing time of job J_i is modified to $p_i' = p_i + t_i$, where t_i is the forced idle time, if any, on the machine immediately preceding J_i and $t_1 = 0$. We call this the *modified problem.*

Remark 5.1 For a given job sequence, modified processing time can be obtained in $O(N)$ time by setting $p'_i = p_i + max\{0, r_i - c_{i-1}\}$, where c_{i-1} is the *completion time* of J_{i-1} in the fixed job sequence, and $c_0 = 0$.

Now we prove that the solution of the modified problem is equivalent to solving the original problem.

Lemma 5.1 *The modified batch scheduling problem is equivalent to the original batch scheduling problem.*

Proof: Consider an $n-$batching schedule with batch sizes $\theta_1, \theta_2, \ldots, \theta_n$. Let the total cost of the optimal schedule of the original problem and the modified problem be TC^* and TC^*_{mod} respectively.

$$\begin{aligned} TC^* &= \sum_{i\in\theta_1} w_i\big(\sum_{j\in\theta_1}(p_j+t_j) - r_i\big) + \sum_{i\in\theta_2} w_i\big(\sum_{j\in(\theta_1\cup\theta_2)}(p_j+t_j) - r_i\big) \\ &\quad + \cdots + \sum_{i\in\theta_n} w_i\big(\sum_{j=1}^{N}(p_j+t_j) - r_i\big) + nd \\ &= \sum_{i\in\theta_1} w_i \sum_{j\in\theta_1}(p_j+t_j) + \sum_{i\in\theta_2} w_i \sum_{j\in(\theta_1\cup\theta_2)}(p_j+t_j) \\ &\quad + \cdots + \sum_{i\in\theta_n} w_i \sum_{j=1}^{N}(p_j+t_j) - \sum_{i=1}^{N} w_i r_i + nd \\ &= \sum_{i\in\theta_1} w_i \sum_{i\in\theta_1} p'_i + \sum_{i\in\theta_2} w_i \sum_{i\in(\theta_1\cup\theta_2)} p'_i + \cdots + \sum_{i\in\theta_n} w_i \sum_{i=1}^{N} p'_i - \sum_{i=1}^{N} w_i r_i + nd \\ &= TC^*_{mod}. \quad \square \end{aligned}$$

Remark 5.2 Let $TC'_{mod} = TC_{mod} - \sum_{i=1}^{N} w_i r_i$. Then it is clear that minimizing TC'_{mod} is equivalent to minimizing TC_{mod} because $\sum_{i=1}^{N} w_i r_i$ is a constant.

Remark 5.3 The modified batch scheduling problem on a fixed job sequence is equivalent to batch scheduling of multiple products without release times. Therefore, Algorithm 3.2 provided in Section 3.4.1 can be used to solve the modified batch scheduling problem on the same fixed job sequence.

Now we provide our optimal batching algorithm for the fixed job sequence problem of the manufacturer.

Algorithm 5.1 (Batching Algorithm for Jobs with Fixed Sequence)

```
Step 1   Input job sequence and other data.
Step 2   Compute the modified processing time for all the jobs.
Step 3   Call Algorithm 3.2 to solve the modified problem.
```

Lemma 5.2 *There exists a polynomial-time algorithm that optimally solves the batch scheduling problem at the manufacturer with fixed job sequence.*

Proof: For any given job sequence, we can find the equivalent modified batching problem in $O(N)$ time. The optimal batching of this modified problem can be found in $O(N)$ time using Algorithm 3.2. □

5.4 Single-Product Batch Scheduling

In this section, we prove that single-product batch scheduling for the manufacturer can be solved polynomially. In the single product problem, all the jobs have identical processing times and identical weights, i.e., $p_i = p$ and $w_i = w$ $(i = 1, 2, \ldots, N)$. We use the following two easy-to-prove properties to prove the existence of a polynomial algorithm.

Property 5.2 *There exists an optimal schedule in which the machine is idle only when no job is available for processing on the machine.*

Proof: Consider a schedule in which the machine idles when there are jobs available for processing. If we start processing any of these available jobs during the machine idle time, the delivery cost of any batch will not be increased. □

Property 5.3 *Since jobs have the same processing times and weights, any job sequence that follows first come (first arrive) first served order is an optimal job sequence.*

Therefore, batching and job sequencing are separable and an optimal job sequence can be obtained in $O(N)$ time.

Lemma 5.3 *There exists a polynomial-time algorithm for the single product batch scheduling problem of the manufacturer.*

Proof: We know that the optimal job sequence can be obtained in polynomial time. From Lemma 5.2, optimal batching of this job sequence can be obtained in $O(N)$ time using Algorithm 5.1. □

For the identical product batching with multiple customers, again we can find the optimal job sequence for each customer from Properties 5.2 and 5.3. Optimal splitting of these customer job sequences can be obtained using the dynamic programming algorithm of Hall and Potts [19] which requires $O(N^{m+1})$ time, where N is the total number of items to be produced and m is the number of customers.

5.5 Batch Scheduling of Jobs with Identical Weight

We study batch scheduling of jobs with release times and arbitrary processings times but with identical (unit) weights, i.e., $w_i = w$ (for $i = 1, 2, \ldots, N$). This problem is NP-hard even if the delivery cost $d = 0$ (Lenstra et. al [25]). Therefore, we develop a 2-approximation algorithm based on the rounding of preempted jobs in an optimal preemptive schedule. The next section analyzes some properties of the optimal preemptive schedule to get the insight on rounding the preempted jobs.

5.5.1 Batch Scheduling in the Preemptive Problem

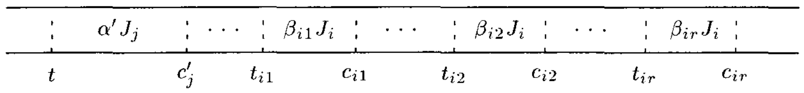
We consider the classical preemptive schedule, where a job can be interrupted while processing and the remaining part is continued later, but the job can be delivered *only* after the whole job is completed. Thus, our objective is to minimize $TC_{pmt} = w\sum_{j=1}^{n}(T_j - r_j) + nd$, where T_j is the delivery time of the jth batch from the manufacturer to the customer. We interchangeably use θ_j to denote the jth batch and the set of jobs in the jth delivery batch to the customer and denote the minimum value of TC_{pmt} by TC^*_{pmt}.

Before developing the approximation algorithm, we first analyze some basic properties of the optimal preemptive schedule to get the insight on rounding the preempted jobs. Note that Properties 5.1 and 5.2 are true for preemptive batch scheduling problems too.

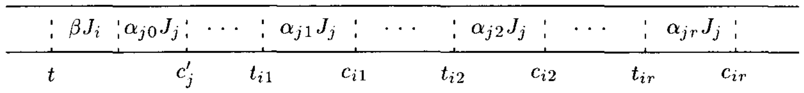
Lemma 5.4 *There exists an optimal schedule in which whenever a job is completed or a new job arrives, the job with the shortest remaining processing time (SRPT) is scheduled next.*

Proof: Let us assume that the lemma does not hold. Then there exists an optimal schedule in which at some time point t, $\alpha' J_j$ is scheduled, where βJ_i and αJ_j, are two of the fractional jobs available for processing at t, and $\beta p_i < \alpha p_j$, $(0 < \alpha' \leq \alpha \leq 1$ and $0 < \beta \leq 1)$.

Let us assume that βJ_i is split into r fractional jobs $\beta_{ik} J_i$ $(k = 1, 2, \ldots, r)$ (refer to Figure 5.2 - a). Let $\beta_{ik} J_i$ start processing at time t_{ik} $(k = 1, 2, \ldots, r)$ and



a- Figure for Lemma 5.4 before exchanging $\alpha' J_j$ and βJ_i.



b- Figure for Lemma 5.4 after exchanging $\alpha' J_j$ and βJ_i.

Figure 5.2: Figure for Lemma 5.4 Case 1.

complete at time c_{ik} in the optimal preemptive schedule. Also assume that $\alpha' J_j$ completes processing at c'_j.

Case 1 $\alpha' p_j \geq \beta p_i$.

Bring $\beta_{i1} J_i, \beta_{i2} J_i, \ldots, \beta_{ir} J_i$ together and schedule the fractional job βJ_i at time t. Also split the processing time of $\alpha' p_j$ into pieces so that they 'fit' into the processing slots previously occupied by the $\beta_{i1} J_i, \beta_{i2} J_i, \ldots, \beta_i r J_i$. Schedule the corresponding $\alpha_{jk} J_j$ $(k = 1, 2, \ldots, r)$ of $\alpha' J_j$ at time $t_{i1}, t_{i2}, \ldots, t_{ir}$ (refer to Figure 5.2 - b), where $\alpha_{jk} = \frac{\beta_{ik} p_i}{p_j}$, and if $\alpha' p_j > \beta p_i$ then schedule $\alpha_{j0} = \frac{\alpha' p_j - \beta p_i}{p_j}$ of J_j right after βJ_i.

We keep batch delivery times the same after the exchange. The number of jobs which complete processing by t remains the same, may increase by 1 in the interval (t, c_{i1}), and after c_{i1} remains the same. Since the delivery times are not changed, at least the same number of jobs will be available for delivery at these times and the sum of the completion times is not increased.

Case 2 $\alpha' p_j < \beta p_i$.

Again refer to Figure 5.2- a. Bring the first available $\beta' = \frac{\alpha' p_j}{p_i}$ fraction of J_i from $\beta_{i1} J_i, \beta_{i2} J_i, \ldots, \beta_{ir} J_i$ together, and assign $\beta' J_i$ at time t and $\alpha' J_j$ in the corresponding time slots of $\beta' J_i$. This will not increase the sum of the completion times, if we do not change the original batch delivery times.

Since the number of batches does not increase in the above two cases, the job exchanges do not change the total delivery cost. Therefore in both cases, the total cost is not increased. Repeating the above exchanges for every violation of the SRPT order will yield a schedule which satisfies the lemma. □

In the optimal preemptive batch schedule, θ_l ($l = 1, 2, \ldots, n$) may contain two types of jobs:

(i) set of jobs which are completely processed in the time interval $(T_{l-1}, T_l]$. We denote this set by F_l.

(ii) set of jobs for which only a fraction of the job is processed in $(T_{l-1}, T_l]$. We denote this set by P_l.

Therefore, $\theta_l = \{F_l \cup P_l\}$. For each job in P_l, the corresponding remaining parts, the *early parts* are processed before T_{l-1} and let u_l be the total time required to process these early parts. Clearly, $\sum_{j=2}^{l} u_j \leq T_{l-1}$ for $l = 2, 3, \ldots, n$.

Lemma 5.5 *Consider a batch θ_l. If we schedule all early fractional jobs of P_l at T_{l-1}, then the change in the sum of the weighted completion time, Δ_l, is not greater than $u_l \sum_{j=l}^{n} w|\theta_j|$.*

Proof: Move all the corresponding early parts of P_l to T_{l-1}. Moving these partial jobs to T_{l-1} will not increase the delivery times of batches $\theta_1, \theta_2, \ldots, \theta_{l-1}$. However, it may increase the delivery times of batches $\theta_l, \theta_{l+1}, \ldots, \theta_n$ by at most u_l time units. Therefore,

$$\Delta_l \leq u_l \sum_{j=l}^{n} w|\theta_j|. \quad \square$$

Lemma 5.6 *Let the change in the total cost due to moving all the early fractional jobs of P_l to T_{l-1} (for $l = 2, 3, \ldots, n$) be Δ. Then $\Delta < TC^*_{pmt}$.*

Proof:

$$\begin{aligned}
\Delta &= \sum_{l=2}^{n} \Delta_l = \sum_{l=2}^{n} (u_l \sum_{j=l}^{n} w|\theta_j|) \\
&= u_2 w(|\theta_2| + |\theta_3| + \cdots + |\theta_n|) + u_3 w(|\theta_3| + |\theta_4| + \cdots + |\theta_n|) \\
&\quad + \cdots + u_n w(|\theta_n|) \\
&= u_2 w|\theta_2| + (u_2 + u_3) w|\theta_3| + \cdots + (u_2 + u_3 + \cdots + u_n) w|\theta_n|
\end{aligned}$$

But $\sum_{j=2}^{i} u_j < T_{i-1}$, for $i = 2, 3, \ldots, n$. Therefore,

$$\begin{aligned}
\Delta &\leq T_1 w|\theta_2| + T_2 w|\theta_3| + \cdots + T_{n-1} w|\theta_n| \\
&< \sum_{i=1}^{n} w|\theta_i| T_i + nd \\
&= TC^*_{pmt} \quad \square
\end{aligned}$$

5.5.2 Non-preemptive Batch Scheduling

We develop the non-preemptive batch schedule using the properties of the optimal preemptive batch schedule of Lemma 5.4. After we moved all the early fractional jobs of P_l to start their processing at T_{l-1} (for $l = 2, 3, \ldots, n$), we obtain a schedule in which all parts of the jobs delivered in θ_l are scheduled at or after T_{l-1}. It is clear that we can turn this (possibly preemptive) schedule into a nonpreemptive one without any further increase in the batch completion times by scheduling all parts together for every job in the order of their job completion times in the optimal preemptive schedule. This will result in a non-preemptive schedule, in which jobs are sequenced according to their completion times in the SRPT schedule. This schedule can be obtained in $O(N)$ time.

We provide now our 2-approximation algorithm.

Algorithm 5.2 (Approximation Algorithm for Jobs with Identical Weights)

Step 1 Find the optimal SRPT job sequence for the preemptive batching problem.

Step 2 Schedule jobs in the order of their completion time in the optimal preemptive schedule.

Step 3 Call Algorithm 5.1 for the optimal batching of the sequence.

Theorem 5.1 *Algorithm 5.2 is a 2-approximation algorithm requiring $O(N)$ time.*

Proof: The total cost of moving all the early partial jobs of P_l to θ_l (for $l = 1, 2, \ldots, n$) in the optimal preemptive batch-delivery schedule is less than TC^*_{pmt} by Lemma 5.6. Thus bringing together all fractions within a batch into a non-preemptive schedule does not increase the cost further, so the resulting schedule will have a cost of at most $2TC^*_{pmt}$. We also know that the jobs of this non-preemptive batch schedule follow the order of their completion in the optimal preemptive schedule. Therefore any optimal batch-delivery schedule on this sequence will not have a cost higher than $2TC^*_{pmt}$. Algorithm 5.1 in Step 4 finds the optimal batching of this job sequence. Hence the Algorithm 5.2 is a 2-approximation algorithm. It is clear that each step of the algorithm can be implemented in $O(N)$ time. □

5.6 Hybrid Algorithm for the Manufacturer's Problem

In this section, we study batch-delivery schedules of jobs with arbitrary processing times and weights at the manufacturer. It is clear that this problem is NP-hard, since the same problem is NP-hard even with identical weights. We develop a hybrid meta-heuristic algorithm to solve this problem. We have proved that the optimal batching of any job sequence can be obtained in $O(N)$ time. Therefore, we consider this problem in two phases. In Phase 1, we generate job sequences based on a Genetic Algorithm (GA) and in Phase 2, we call Algorithm 5.1 to get the optimal batching of these sequences. Phase 2 is in fact called in Phase 1, since we need the fitness value of each schedule in GA. We explain the procedures of our algorithm in detail in the next section.

5.6.1 Phase 1: Genetic Algorithm

In early application of GA in scheduling, the primary strategy used for encoding sequences was the literal permutation order. In *permutation order encoding*, each chromosome represents the job order in the corresponding schedule. For example, in a permutation order encoding system, a job sequence $\{3, 2, 5, 1, 4\}$ is encoded as given below:

3	2	5	1	4

Whenever a crossover or mutation operation is carried out, the feasibility of the schedule is not guaranteed by this encoding system. Thus, using permutation order encoding forces the algorithm developer to apply specialized operators in the crossover procedure in order to maintain the feasibility. To avoid this difficulty, we use the random keys encoding of Bean [4].

Random Keys Genetic Algorithm (RKGA)

We briefly review the concept of random keys in a genetic algorithm. RKGA is widely used in GA applications for scheduling problems because it only produces feasible offsprings. Moreover, relative and absolute ordering information can be preserved after recombination of parents. In RKGA a chromosome is assigned a string of random numbers. For example, consider the chromosome:

.345	.983	.726	.167	.529

The intepretation of this chromosome is as follows: Jobs $J_1, J_2, \ldots, J_5$ are assigned random numbers 0.345,0.983,0.726,0.167, and 0.529, respectively. Then the job sequence of this chromosome is obtained by ordering the jobs in ascending order of their random numbers. Therefore, the job sequence of this chromosome is $\{4, 1, 5, 3, 2\}$.

We briefly describe the major steps of our GA below.

Initial Population

We use a population size of 30 in our GA. The initial population includes the optimal batch-delivery schedule of the ready WSPT job sequence. (A *ready WSPT* job sequence is a schedule in which at each arrival and/or job completion, a job J_i with the largest $\frac{w_i}{p_i}$ ratio among the available jobs is scheduled next.)

We also include 9 schedules in which jobs in the same block are scheduled consecutively, i.e., jobs in block k are scheduled before those in block l $(l > k)$. These precedence constraints between the jobs in different blocks are easily handled in RKGA by assigning random numbers from increasing and disjoint ranges to jobs in later blocks. For example, let us assume there are 2 blocks formed on 10 jobs. The first 7 jobs form the first block and the next 3 form the second block. Then for the first 7 jobs we generate random numbers from the range (0,10), and for the next 3 jobs we assign random numbers from the range (11,15). This way a random number assigned to any job in the second block will always be greater than that of any job in the first block. Thus the precedence constraint is handled automatically.

Another 20 schedules contain jobs in random order. For this set of schedules, random keys are assigned randomly.

Fitness Evaluation We decode each chromosome into a job sequence and call Phase 2 to obtain the optimal batching of each sequence and its corresponding cost. For each population, we find the maximum cost of all its batch-delivery schedules (maxcost), and for each individual in the population we set $fitnessvalue = \frac{maxcost-cost\ of\ the\ individual}{30*maxcost-sum\ of\ the\ cost\ of\ all\ individuals}$.

Parent Selection We use the roulette wheel method to select two parents to create two new offsprings. In *roulette wheel* selection, the probability of selecting a parent is proportional to its fitness value.

Crossover A two point crossover is used with a crossover probability $p_{cros} = 0.9$.

Mutation For each chromosome obtained after crossover, we select a random integer k not greater than N. Then we randomly select k genes (jobs) of that chromosome and for each selected gene, we assign a new random key with mutation probability of $p_{mut} = 0.1$.

5.6.2 Phase 2: Optimal Batching of Job Sequence

In Phase 2, we call Algorithm 5.1 to get the optimal batching of each individual.

To analyse the performance of our heuristic, we develop a lower bound explained in the following section.

5.6.3 Lower Bound

We use the preemptive schedule to obtain a lower bound for the problem. We make the assumption that jobs arrive from the supplier and are delivered to the customer at discrete time points. Therefore it is clear that in the optimal preemptive batch-delivery schedule, a job is split only at discrete time points. Consider a set of jobs $J_1, J_2, \ldots, J_N$. We replace each job J_i $(i = 1, 2, \ldots, N)$ with p_i unit processing time (UPT) jobs, i.e., J'_k with processing time $p'_k = 1$, weight $w'_k = \frac{w_i}{p_i}$ and release time $r'_k = r_i$ (for $k = \sum_{j=1}^{i-1} p_j + 1, \sum_{j=1}^{i-1} p_j + 2, \ldots, \sum_{j=1}^{i} p_j$). We find the optimal batching of these UPT jobs, and the total cost of this optimal batch-delivery schedule will be a lower bound for our problem. We briefly discuss the optimal batching of UPT jobs.

Optimal Batching of UPT Jobs

If we could find the optimal job sequence for the UPT jobs then Algorithm 5.1

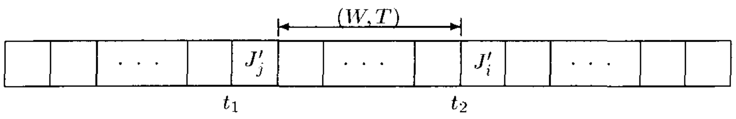


Figure 5.3: Schedule for Lemma 5.7.

will find the optimal batching of the sequence. In Lemma 5.7, we prove that ready LW job sequence is optimal for the UPT problem. A *ready LW* schedule is one in which at each job arrival and/or completion, a job with the largest weight among the available jobs is scheduled next.

Lemma 5.7 *Ready LW schedule is optimal to minimize the total cost of UPT jobs.*

Proof: Let us assume that the lemma does not hold. Then there exists an optimal schedule in which J'_j is scheduled at time t_1 and J'_i at t_2, where $t_2 > t_1$ and $w'_i > w'_j$. Let $t_2 - (t_1 + 1) = T$ (refer to Figure 5.3) and the total weight of all the jobs scheduled in the time interval $[t_1 + 1, t_2]$ be W.

Let us exchange jobs J'_i and J'_j while not affecting the batch delivery times and let the change in the total cost be Δ. Then

$$\begin{aligned}\Delta &= w'_j(T+1) - w'_i(T+1)\\ &= (T+1)(w'_j - w'_i) < 0\end{aligned}$$

This contradicts the optimality assumption. □

Since we know the optimal job sequence for the batching problem, using Algorithm 5.1, we can obtain the optimal batching of this job sequence. Now we describe the major steps of our algorithm to find the lower bound.

Algorithm 5.3 (Batching Algorithm for UPT Jobs)

Step 1 Convert the problem to the equivalent UPT problem.

Step 2 Schedule UPT jobs in ready LW order.

Step 3 Call Algorithm 5.1 to find the optimal batching of the ready LW schedule.

5.7 Computational Experiment

In order to analyze the performance of our hybrid meta-heuristic algorithm, we coded it and the lowerbound algorithm in ANCII C and tested it on a SUN computer. The results of our experiments are summarised in Tables 5.1 and 5.2. We tested the algorithm for 50 jobs with processing times randomly generated from $U(1,100)$ (uniform distribution). Jobs arrive in batches at different time points. Batch sizes are randomly generated from $U(1,5)$ and arrival times are from $U(0, 50.5 * 50 * R)$. We set 6 different values for R and therefore, there are 6 instance types with $R = 0.2$, $R = 0.4$, $R = 0.6$, $R = 0.8$, $R = 1.0$ and $R = 2.0$ respectively. We consider 3 different delivery costs $d = 500$, $d = 1000$, and $d = 5000$ for each instance. We ran the algorithm for 10 randomly generated problems for each treatment and obtained the *percentage gap* $= \frac{hybrid\ solution\ - lowerbound}{lowerbound} * 100$. In Table 5.1, weights are from $U(1,100)$ whereas in Table 5.2 weights are from $U(1,10)$. The results of the experiments show that our algorithm's solution is close to the lowerbound, as the maximum gap was about 5%.

The CPU time is measured in seconds. The tables show that in almost all the cases, the average percentage gap decreases with increasing batch delivery cost. This may be because the larger delivery cost results in larger batch sizes and job sequence within a batch does not affect the total cost. Further the results show that the average percentage gap increases when increasing R from 0.2 and then decreases when R becomes higher.

Remark 5.4 We could also use the above algorithm for the multiple customer batch scheduling problem at the manufacturer, with a minor modification. The algorithm by Hall and Potts [19] can be used to get the optimal batching of given job sequence. Therefore, in Step 3, we would have to call the dynamic programming algorithm by

Instance Type	$d = 500$			$d = 1000$			$d = 5000$		
	Avg. CPU Time	% Gap		Avg. CPU Time	% Gap		Avg. CPU Time	% Gap	
		Avg.	Max		Avg.	Max		Avg.	Max
$R = 0.2$	6.0	1.940	2.930	6.3	2.038	3.833	7.3	1.680	4.399
$R = 0.4$	5.9	2.723	4.397	6.4	2.054	5.008	7.2	1.596	2.161
$R = 0.6$	6.0	2.716	3.811	6.3	2.410	4.248	7.1	2.328	3.933
$R = 0.8$	5.8	2.447	3.435	6.2	1.826	4.157	7.2	1.543	2.905
$R = 1.0$	5.8	1.615	2.498	6.2	1.354	2.686	7.0	1.133	1.954
$R = 2.0$	5.7	0.751	1.229	6.1	0.452	1.090	7.4	0.329	0.741

Table 5.1: Performance of Algorithm 5.2 for $w_i = U(1, 100)$, $p_i = U(1, 100)$

Instance Type	$d = 500$			$d = 1000$			$d = 5000$		
	Avg. CPU Time	% Gap		Avg. CPU Time	% Gap		Avg. CPU Time	% Gap	
		Avg.	Max		Avg.	Max		Avg.	Max
$R = 0.2$	7.0	1.773	3.549	7.2	1.297	3.430	9.0	1.145	2.156
$R = 0.4$	7.0	2.377	4.900	7.4	1.581	2.664	9.2	1.311	2.377
$R = 0.6$	7.2	1.708	3.544	7.8	1.460	2.444	9.2	1.573	4.327
$R = 0.8$	7.0	1.657	2.722	7.7	1.573	2.753	9.2	1.144	1.640
$R = 1.0$	7.0	0.856	1.330	7.7	1.343	2.958	8.8	0.728	1.244
$R = 2.0$	6.9	0.359	0.937	8.8	0.293	0.646	8.8	0.262	0.441

Table 5.2: Performance of Algorithm 5.2 for $w_i = U(1, 10)$, $p_i = U(1, 100)$

Hall and Potts.

Chapter 6

Batch Scheduling in Customer Centric Supply Chains

6.1 Introduction

This chapter studies batch scheduling at the manufacturer in a customer centric supply chain. There are N jobs, $J_1, J_2, \ldots, J_N$, to be processed at the manufacturer whose system may be modeled by either a single machine or an assembly-type operation with subtasks $J_{i,j}$ to be processed on l machines in a series for $i = 1, ..., N$ and $j = 1, ..., l$. Job J_i must be delivered to a customer at time D_i. The cost of these deliveries is borne by the customer. In the single-machine model, J_i requires processing for p_i time for $i = 1, 2, \ldots, N$. In the assembly operation, the processing time of subtask $J_{i,j}$ is denoted by $p_{i,j}$. (If a job skips a certain operation then $p_{i,j} = 0$ for the corresponding subtask.) Since no job is delivered before its deadline, the manufacturer wants to complete them as close to these deadlines as possible. Therefore, we assume that the jobs are processed in earliest due date (EDD) order and this leads to a feasible schedule, i.e., the manufacturer has sufficient capacity to make this schedule feasible for meeting the deadlines. The manufacturer receives parts and supplies for each job or subtask from his supplier(s) in batches and is charged a delivery cost of d for each batch. The manufacturer must receive the batches in time to enable him to meet the final deadlines, but does not want to receive the supplies too early because each job J_i incurs an inventory holding cost in the time interval $[a_i, D_i]$, where a_i is its *arrival time* at the manufacturer for $i = 1, 2, \ldots, N$. The inventory holding cost of a job J_i is closely related to its *flow time* defined as $D_i - a_i$. Since the delivery cost is measured in monetary terms, we multiply flow-time related performance measures by appropriate constants in order to maintain compatibility in measurement. Therefore, we multiply the sum of flow times by a constant w, which is the cost of holding a job in inventory over a unit time; multiply the maximum flow time by a constant K, which is the penalty cost associated with the maximum flow time; and multiply the flow time of job J_i by w_i, which is the holding cost of job J_i over a time unit when the objective is to minimize the sum of the weighted flow times and delivery costs. The manufacturer wants to find the optimal *arrival time* a_j of each job J_j, the *number of batches* n and the partitioning of the jobs into arrival batches so that the total cost is minimized. We consider the following objectives:

1. For the sum of flow times with batching, total cost $TC_1 = \sum_{j=1}^{N} w(D_j - a_j) + nd$;
2. For the maximum flow time with batching, total cost $TC_2 = K \max_{j=1,\ldots,N}(D_j - a_j) + nd$;
3. For the sum of weighted flow times with batching, total cost $TC_3 = \sum_{j=1}^{N} w_j(D_j - a_j) + nd$.

The chapter proceeds as follows. In the next section, we study problems of batch arrival scheduling to minimize the total weighted flow time and delivery costs, i.e., cost function TC_3. First we prove that the problem is strongly NP-hard on a single machine even with a common due date for all the jobs. Following this, we present a linear-time dynamic programming algorithm for the problem on a *fixed* job arrival sequence. This algorithm is used repeatedly in Section 6.3 for minimizing TC_1 both for single-machine and assembly-shop environments. In Section 6.4, we present an efficient dynamic programming algorithm for batch arrival scheduling with objective TC_2.

6.2 Batch Arrival Scheduling to Minimize the Total Weighted Flow Time and Delivery Costs

6.2.1 Complexity

Let us consider the batch arrival scheduling problem at the manufacturer when its system is modeled by a single machine and the objective is to minimize TC_3.

Theorem 6.1 *Minimizing* $TC_3 = \sum_{j=1}^{N} w_j(D_j - a_j) + nd$ *is strongly NP-hard.*

Proof: Hall and Potts [19] have proved that minimizing the sum of total weighted flow times and delivery costs for a supplier in a push-type system is strongly NP-hard. They used the well-known strongly NP-hard 3-PARTITION problem to reduce it to their scheduling problem. We show how this reduction can be adapted to prove the strong NP-hardness of our problem.

3-PARTITION [18]:

Given $3r$ integers $u_1, ..., u_{3r}$, where $\sum_{i=1}^{3r} u_i = rz$ and $z/4 < u_i < z/2$, for $i = 1, ..., 3r$, does there exist a partition $A_1, ..., A_r$ of the index set $\{1, ..., 3r\}$, such that $|A_j| = 3$ and $\sum_{i \in A_j} u_i = z$, for $j = 1, ..., r$?

Consider the following instance of our scheduling problem: $N = 3r$, job J_i has $p_i = w_i = u_i$ and $D_i = rz$, for $i = 1, ..., 3r$, $d = z^2/2$ and let $C = r(r+2)z^2/2$

be a threshold value. We prove that there exists a batch arrival schedule for this instance with $TC_3 = \sum_{j=1}^{N} w_j(D_j - a_j) + nd \leq C$ if and only if there exists a solution for 3-PARTITION.

Suppose 3-PARTITION has a solution and assume, without loss of generality, that the integers are numbered so that $u_{3i-2}+u_{3i-1}+u_{3i} = z$, for $i = 1, ..., r$. Consider the schedule in which the jobs are scheduled in the sequence $J_1, ..., J_N$ and supply batch B_i for jobs $\{J_{3i-2}, J_{3i-1}, J_{3i}\}$ arrives at time J_{3i-2} and starts its processing right on, i.e., $a_{3i-2} = a_{3i-1} = a_{3i} = (i-1)z$, for $i = 1, ..., r$. It is easy to see that the jobs $\{J_{3i-2}, J_{3i-1}, J_{3i}\}$ have flow time equal to $rz - (i-1)z = (r-i+1)z$. Therefore, $TC_3 = \sum_{i=1}^{r}(u_{3i-2} + u_{3i-1} + u_{3i})(r-i+1)z + rd = \sum_{i=1}^{r}(r-i+1)z^2 + rz^2/2 = C$ for this schedule.

Next we prove the theorem in the other direction. Suppose we have a schedule with n arrival batches and let x_i be the total processing time of the jobs corresponding to the ith batch B_i for $i = 1, ..., n$. It is clear that supplies for B_i must arrive at the time the last job in B_{i-1} completes its processing, i.e., at $\sum_{j=1}^{i-1} x_j$. Therefore, the flow time of the jobs in B_i will be $rz - \sum_{j=1}^{i-1} x_j = \sum_{j=i}^{n} x_j$ for $i = 1, ..., n$. Thus we have $TC_3 = \sum_{i=1}^{n} x_i \sum_{j=i}^{n} x_j + nd = \sum_{i=1}^{n} x_i \sum_{j=i}^{n} x_j + nz^2/2 = \left(\sum_{j=1}^{n} x_j\right)^2/2 + \sum_{j=1}^{n} x_j^2/2 + nz^2/2$. Thus minimizing TC_3 on n batches can be written as

$$minimize \left(\sum_{j=1}^{n} x_j\right)^2/2 + \sum_{j=1}^{n} x_j^2/2 + nz^2/2$$

subject to

$$\sum_{j=1}^{n} x_j = rz.$$

Since the first term of this objective is $(rz)^2$, it is easy to see that the whole function will be minimized when $x_1 = x_2 = ... = x_n = rz/n$. Thus for any n-batch solution we must have $TC_3 \geq (rz)^2/2 + n(rz/n)^2/2 + nz^2/2$. Simple arguments from calculus show that this expression reaches its minimum at $n = r$ and the minimum value is C. Thus if there exists a batching schedule with $TC_3 = C$, then we must have $n = r$ and each batch must have a size $x_j = z$. This implies that each batch has 3 jobs in it and 3-PARTITION has a solution. $\square$

6.2.2 Batching a Given Job Sequence on a Single Machine

In this section, we study the optimal batching problem at the manufacturer to minimize $TC_3 = \sum_{j=1}^{N} w_j(D_j - a_j) + nd$ when the order of job processing at the manufacturer is given and this is also the order of job arrivals. Without loss of generality, let this sequence be $J_1, ...J_N$. Note that the given job processing sequence is assumed to be feasible for meeting the promised delivery times. Let S_i denote the *latest start time* for job J_i such that the schedule is feasible.

Consider an n-batch arrival schedule, and let i_j be the index of the first job of arrival batch B_j, i.e., the batch schedule is $\{i_1, i_1+1, \ldots, i_2-1\}, \{i_2, i_2+1, \ldots, i_3-1\}, \ldots, \{i_n, i_n+1, \ldots, N\}$. Then it is easy to see that batch B_j should arrive at time S_{i_j} and not earlier. Thus

$$TC_3 = \sum_{k=1}^{n} \sum_{j=i_k}^{i_{k+1}-1} w_j(D_j - S_{i_k}) + nd.$$

If we define $S_{N+1} = D_N$, then we can write $S_{i_k} = D_N - \sum_{j=i_k}^{N} (S_{j+1} - S_j)$. Therefore,

$$\begin{aligned} TC_3 &= \sum_{j=1}^{N} w_j D_j - \sum_{k=1}^{n} \sum_{j=i_k}^{i_{k+1}-1} w_j\Big(D_N - \sum_{j=i_k}^{N} (S_{j+1} - S_j)\Big) + nd \\ &= \sum_{j=1}^{N} w_j(D_j - D_N) + \sum_{k=1}^{n} \sum_{j=i_k}^{i_{k+1}-1} w_j \sum_{j=i_k}^{N} (S_{j+1} - S_j) + nd. \end{aligned}$$

Let $S_{j+1} - S_j = p'_j$ for $j = 1, 2, \ldots, N$. Note that $p'_j \geq p_j$ and p'_j can be interpreted as the length of time on the machine 'allocated' to job j. (We have $p'_j > p_j$ if there is an idle time in the schedule.) Then

$$TC_3 = \sum_{j=1}^{N} w_j(D_j - D_N) + \sum_{k=1}^{n} \sum_{j=i_k}^{i_{k+1}-1} w_j \sum_{j=i_k}^{N} p'_j + nd.$$

$$\begin{aligned} TC_3 &= \sum_{j=1}^{N} w_j(D_j - D_N) + \sum_{k=1}^{n} \Big(\sum_{j=i_k}^{i_{k+1}-1} w_j \sum_{j=i_k}^{N} p'_j \Big) + nd \\ &= \sum_{j=1}^{N} w_j(D_j - D_N) + \sum_{k=1}^{n} \Big(\sum_{j=i_k}^{N} p'_j \sum_{j=i_k}^{i_{k+1}-1} w_j + d \Big) \end{aligned}$$

Therefore minimizing TC_3 can also be formulated as a special shortest path problem by exchanging processing times for weights and weights for allocated times p'_j in our formulations discussed in Section 3.4.1. Thus the Algorithm 3.2 given in Section 3.4.1 will find the optimal batching solution.

Theorem 6.2 *The optimal batching which minimizes* $\sum_{j=1}^{N} w_j(D_j - a_j) + nd$ *on a* ***given*** *job sequence can be found in* $O(N)$ *time.*

6.3 Minimizing the Total Flow Time and Delivery Costs

In this section, we study simultaneous sequencing and batching of jobs for arrival at the manufacturer to minimize $TC_1 = (\sum_{j=1}^{N} w(D_j - a_j) + nd)$.

6.3.1 Scheduling Batch Arrivals on a Single Machine

Lemma 6.1 *There is an optimal schedule in which the order of job arrivals is the same as the order of job processing.*

Proof: A job cannot start processing until the job immediately preceding it in the EDD processing sequence is not completed, and the arrival of any job before the arrival of a job preceding it in the processing sequence can only make the total flow time larger. Therefore, no job should arrive in the optimal schedule before any of its predecessors in the EDD order. □

Without loss of generality, we index jobs in the order they are in this common sequence of arrival and processing. Then the latest possible start time of job J_i can be recursively calculated by $S_N = D_N - p_N$, and $S_i = \min\{D_i, S_{i+1}\} - p_i$ for $i = N-1, N-2, ..., 2, 1$.

Although the job arrival and job processing sequence is the same, the optimal schedule may contain jobs which arrive early and wait in the shop. Consider the following example:

j	1	2	3	4	5	6
p_j	9	7	5	12	6	5
D_j	23	23	23	48	48	48

Let us say that we are given the job processing sequence {1,2,3,4,5,6} and we have to find the optimal 2-batching schedule. For this problem, the optimal batching is $B_1 = \{1, 2, 3, 4\}$ and $B_2 = \{5, 6\}$ with total flow time of 131. In this batching solution, the second batch arrives at $t = S_5 = 37$ and the processing of job set $\{5, 6\}$ is completed at $t = 48$; the first batch arrives at $t = S_1 = 2$ and the processing of job set $\{1, 2, 3\}$ is completed at $t = 23$, and the processing of job set $\{4\}$ is completed at $t = 35$. If we move job J_4 to the second batch, then B_2 must arrive at $t = 25$ so that the job set $\{4, 5, 6\}$ will be completed by $t = 48$; and the job set $\{1, 2, 3\}$ must arrive at $t = 2$ and will be completed at time $t = 23$. The total flow time of this new schedule will increase to 132. This shows that in the optimal schedule some jobs may arrive with early batches and wait in the shop.

Lemma 6.2 *There exists an optimal schedule in which a batch arrives only when all the previously available jobs at the manufacturer have completed processing, i.e., a batch arrives only when the machine is available to start its processing.*

Proof: Since the job processing sequence is given, a job cannot be started before its immediately preceding job is completed. From Lemma 6.1, the order of job arrivals follows the processing sequence. Therefore, jobs assigned to any batch will be processed after the last job of the previous batch has completed processing. Thus arrival of a batch before the completion of the processing of the last job of the previous arrival batch can only increase the flow time of the schedule. □

Lemma 6.3 *Let job J_j be the first job to be processed in arrival batch B_k, then batch B_k should arrive at time S_j.*

Proof: Let us assume that there is an optimal schedule which does not satisfy the lemma, but it is consistent with Lemmas 6.1 and 6.2. Select the last batch which arrives before the latest start time of the first job of the batch in this schedule. If we delay the batch arrival time to the latest start time of the first job of this batch, the schedule will remain feasible. Furthermore, the flow times of all the jobs belonging to this batch will be reduced, flow times of all the jobs belonging to other batches will remain the same. This contradicts the optimality assumption for the schedule. □

Lemma 6.4 *There exists an optimal schedule in which the jobs to be delivered to customer(s) at the same due date are scheduled in LPT order at the manufacturer.*

Proof: We know that jobs to be delivered at the same delivery time to customer(s) are processed consecutively at the manufacturer because of the EDD processing order. Without loss of generality, let this sequence be $J_1, ..., J_N$. Let us assume that the lemma is not true for an optimal schedule. Then there will be at least two jobs J_i and J_{i+1} at the manufacturer such that $p_i < p_{i+1}$ and $D_i = D_{i+1}$. If jobs J_i and J_{i+1} belong to the same arrival batch, then interchanging them will not affect the flow time of any job. If jobs J_i and J_{i+1} belong to different arrival batches, say, to B_k and B_{k+1}, respectively, then let the arrival time of batch B_k be t_k and the arrival time of B_{k+1} be t_{k+1}. From Lemma 6.3, we know $t_{k+1} = min\{D_{i+1}, S_{i+2}\} - p_{i+1}$. Interchange jobs J_i and J_{i+1} in these batches, which does not change the arrival batch sizes. Call the new batches $B_k^{'}$ and $B_{k+1}^{'}$. Thus in the new schedule, $B_k^{'}$ arrives at t_k and $B_{k+1}^{'}$ arrives at $t'_{k+1} = min\{D_i, S_{i+2}\} - p_i = min\{D_{i+1}, S_{i+2}\} - p_i = t_{k+1} + p_{i+1} - p_i$. Note that the interchange will not affect the feasibility of the schedule. The interchange will not affect the flow times of the jobs in $B_k \backslash J_i$ in the original schedule. The flow time of every job in $B_{k+1} \backslash J_{i+1}$ is decreased by $p_{i+1} - p_i > 0$ compared to the original schedule. The flow time of J_{i+1} is increased by $t_{k+1} - t_k$ and the flow time of J_i is decreased by $t'_{k+1} - t_k > t_{k+1} - t_k$. Thus the net change in the total flow time is a decrease by at least $p_{i+1} - p_i$, which contradicts the optimality of the original schedule. Therefore, any jobs J_i and J_{i+1} not in LPT order must belong to the same batch. Repeatedly resequencing the jobs with the same due date into LPT order within the batches does not change the cost or the feasibility of the schedule and leads to an optimal schedule satisfying the conditions of the lemma. □

The combination of EDD ordering with LPT ordering of jobs with the same due date within batches fixes the optimal sequence for the jobs. Since we know the job sequence, Algorithm 3.2 given in Section 3.4.1 can be used to find the optimal batch sizes. Algorithm 6.1 summarizes the steps needed to find the optimal batch arrival schedule.

Theorem 6.3 *Algorithm 6.1 finds in $O(N \log N)$ time an optimal batch arrival*

Algorithm 6.1: *Algorithm to minimize the sum of flow times and delivery costs*

```
Step 1:  Order the jobs in EDD order and schedule the jobs with the same due date in
         LPT order.
Step 2:  Call Algorithm 3.2 to find the optimal arrival batch sizes of the sequence found.
```

schedule that minimizes TC_1, the sum of flow times and delivery costs.

Proof: Step 1 finds the optimal job sequence by sorting, which requires $O(N \log N)$ time. Algorithm 3.2 finds the optimal batching of this job processing sequence in $O(N)$ time. □

6.3.2 Batch Arrival Scheduling for an Assembly Shop

In this section, we study the optimal batch arrival policy in an assembly shop where jobs are processed and assembled on a series of l machines. At each machine, a job may require parts which are delivered from one of q suppliers. The schematic of the supply chain for this problem is shown in Figure 6.1. The manufacturer has to deliver the right products in the right quantities at the promised times to customers and delivery costs are charged to the customers. In order to meet the promised delivery times, the manufacturer has to order the parts from the suppliers, and process and assemble them on the l machines. For any product J_i $(i = 1, 2, \ldots, N)$ which does not need processing on the jth machine $(j = 1, 2, \ldots, l)$, we set $p_{i,j}$ to zero. Suppliers have to deliver parts to the manufacturer at the manufacturer's required times and the costs for deliveries from part suppliers to the manufacturer are charged to the manufacturer. Thus the manufacturer wants to find the optimal batch arrival schedules for the parts from each supplier so that the total of sum of flow times and delivery costs is minimized while meeting promised delivery times to customers.

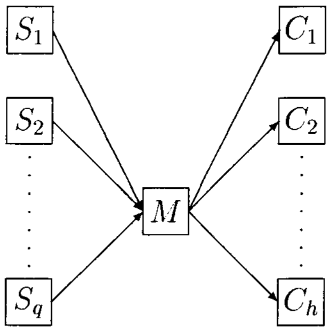


Figure 6.1: Network showing the manufacturer's relationship with q suppliers, and h customers

Let $S_{i,j}$ denote the *latest possible start time* of task $J_{i,j}$ in a feasible schedule. Then the fact that the jobs are 'pulling' their subtasks through the system can be captured by the following backward recursive calculations:

$$\left.\begin{array}{l} S_{N,l} = D_N - p_{N,l} \\ S_{i,l} = min\{D_i, S_{i+1,l}\} - p_{i,l}, \text{ for } i = 1,2,...,N-1 \\ S_{i,j} = min\{S_{i,j+1}, S_{i+1,j}\} - p_{i,j}, \text{ for } i = 1,2,...,N\text{: } j = 1,2,...,l-1 \end{array}\right\} \quad (6.1)$$

Lemma 6.5 *There exists an optimal batch arrival schedule and associated production schedule in which task $J_{i,j}$ (for $i = 1,2,...,N$; $j = 1,2,...,l$) starts its processing at time $S_{i,j}$.*

Proof: By using $S_{i+1,l}$ as an upper bound on the completion time of task $J_{i,l}$, the first two rows of (6.1) ensure that sufficient time will be available at the last machine to finish the processing of $J_{i,l}$ by D_i. The calculations in the last row of (6.1) make sure that there is sufficient time also for task $J_{i,j}$ at machine j for $i = 1,2,...,N$; $j = 1,2,...,l$. (The schedule is feasible if $S_{i,1} \geq 0$ for $i = 1,2,...,N$.) It is also clear that the *arrival time of the parts* for $J_{i,j}$, $a_{i,j}$, must satisfy $a_{i,j} \leq S_{i,j}$ for the schedule to be feasible, but some parts may arrive early. Now suppose we have an optimal schedule in which there are some tasks starting before their latest start time. Consider the last such task, say $J_{r,k}$, and shift its processing to start at $S_{r,k}$. The shift will neither affect the feasibility of the schedule nor the flow time of any task. Repeatedly applying the above argument to all remaining early tasks will lead to a schedule which satisfies the lemma. □

Lemma 6.6 *The batch arrival scheduling problems from each supplier are separable and can be solved independent of each other.*

Proof: We know from Lemma 6.5 that there exists an optimal production schedule in which each task $J_{i,j}$ starts at its latest possible start time $S_{i,j}$. Then $S_{i,j}$ can be viewed as the deadline for the arrival of the parts needed from their supplier. Since each task $J_{i,j}$ receives its part(s) from at most one supplier by assumption, each $S_{i,j}$ can become a delivery deadline only for one supplier. Thus whatever batch arrival times are scheduled from a supplier, this does not affect the flow time of other parts (tasks) from other suppliers. So by considering the delivery requirements from one supplier, we get a separable batch arrival scheduling problem for this supplier. Therefore, the problems can be solved separate from each other for each supplier. □

Theorem 6.4 *The batch arrival scheduling problem at an assembly manufacturer can be optimally solved in $O(qlN\log(lN))$ time.*

Proof: By Lemma 6.6, parts arriving from each supplier can be scheduled for arrival in a separate batch scheduling problem. We have (at most) q of these problems. Each of them can be solved by Algorithm 6.1 in $O(lN\log(lN))$ time, thus the overall time required is at most $O(qlN\log(lN))$. □

6.4 Minimizing Maximum Flow Time and Delivery Costs

In this section, we consider the batch arrival scheduling problem with objective $TC_2 = K \max_{j=1,2,\ldots,N}\{D_j - a_j\} + nd$ at the manufacturer. It is easy to see that Lemmas 6.1-6.3 apply to this problem too and they can be proved the same way.

Lemma 6.7 *There exists an optimal schedule in which the jobs to be delivered to customer(s) at the same due date are scheduled in LPT order at the manufacturer.*

Proof: Let there be an optimal schedule in which jobs with the same due date do not follow the LPT order. Without loss of generality, let the job sequence be $J_1, J_2, \ldots, J_N$. Then there will be at least two jobs J_i and J_{i+1} with $p_i < p_{i+1}$ and $D_i = D_{i+1}$. If J_i and J_{i+1} belong to the same batch, then interchanging these two jobs will not affect the maximum flow time. If jobs J_i and J_{i+1} belong to different arrival batches, say, to B_k and B_{k+1}, respectively, then let the arrival time of batch B_k be t_k and the arrival time of B_{k+1} be t_{k+1}. From Lemma 6.3, we know $t_{k+1} = min\{D_{i+1}, S_{i+2}\} - p_{i+1}$. Interchange jobs J_i and J_{i+1} in these batches without changing the arrival batch sizes. Call the new batches B_k' and B_{k+1}'. Thus in the new schedule, B_k' arrives at t_k and B_{k+1}' arrives at $t_{k+1}' = min\{D_i, S_{i+2}\} - p_i = min\{D_{i+1}, S_{i+2}\} - p_i = t_{k+1} + p_{i+1} - p_i$. Note that the interchange will not affect the feasibility of the schedule. Furthermore, the interchange will not affect the flow times of the jobs in $B_k \backslash J_i$ in the original schedule. The flow time of every job in $B_{k+1} \backslash J_{i+1}$ is decreased by $p_{i+1} - p_i > 0$ compared to the original schedule and the flow time of J_i clearly decreases. The only flow time that is increased is that of J_{i+1}, which goes up by $t_{k+1} - t_k < D_{i+1} - t_k$. We have, however, $D_{i+1} - t_k = D_i - t_k$, and the latter is the flow time of J_i in the original schedule. Therefore, the maximum flow time of the new schedule will not be greater than that of the original one. Repeating this interchange for every violation of the job order in the lemma will yield an optimal schedule satisfying its conditions. □

Note that the lemma implies that there is a job sequence which is optimal for both the maximum flow time plus delivery cost and sum of flow time plus delivery cost objectives. To find the optimal arrival batching for $TC_2 = K \max_{j=1,2,\ldots,N}\{D_j - a_j\} + nd$, however, we cannot use Algorithm 3.2, which was designed for the sum of flow times objective. Therefore, we present a new dynamic programming algorithm below.

Dynamic Programming Algorithm: *Algorithm to minimize TC_2*

Let $f(k,j)$ be the minimum value of TC_2 on the first j jobs in a schedule using k arrival batches for $1 \le k \le j \le N$. For easier notation, we also define $f(k,j) = \infty$ for $1 \le j < k \le N$. The optimal value of TC_2 can be obtained by $\min_{k=1,\ldots,N} f(k,N)$. The recursive computation of $f(k,j)$ for $1 \le k \le j \le N$ is defined as follows.

$$f(k,j) = \min \begin{cases} K \min_{r=k-1,\ldots,j-1} \left\{ \max\{[f(k-1,r) - (k-1)d]/K, \max_{u=r+1,\ldots,j} \{D_u - S_{r+1}\} \right\}\} + kd \\ \\ K \max\left\{[f(k,j-1) - kd]/K, D_j - S_{i(k,j-1)}\right\} + kd, \end{cases}$$

where $S_{i(k,j-1)}$ is the starting time of the first job, $i(k,j-1)$, of the last batch in the schedule realizing $f(k,j-1)$. The first row of the recursion corresponds to taking the optimal schedule realizing $f(k-1,r)$ and adding to it a new arrival batch containing jobs $\{r+1,\ldots,j\}$ for $r = k-1,\ldots,j-1$. Here $[f(k-1,r) - (k-1)d]/K$ expresses the maximum flow time of the schedule realizing $f(k-1,r)$. The second row of the recursion corresponds to the case when job J_j is simply added to the last batch of the schedule realizing $f(k,j-1)$ without starting a new batch. To facilitate the computations, we need to store the index of the first job of the last batch in the schedule realizing $f(k,j)$, denoted by $i(k,j)$.

Initial conditions: $f(0,0) = 0$ and $f(k,j) = \infty$ for $j,k = 1,\ldots,N$.

Theorem 6.5 *Algorithm 6.2 finds an optimal batch arrival schedule at the manufacturer to minimize the total cost of maximum flow time and deliveries in* $O(N^3)$ *time.*

Proof. The algorithm needs to compute $O(N^2)$ $f(k,j)$ values. Each computation needs $O(N)$ time. By storing the indices $i(k,j)$, we can obtain the optimal batching at the end by backtracking. □

Chapter 7

Supplier-Manufacturer Coordination and Batch Scheduling

7.1 Introduction

In Chapters 3 to 6, we studied different batch scheduling problems at the supplier and manufacturer without considering the coordination between players. In this chapter, we treat the supplier and the manufacturer as a single system and study the coordination and batch scheduling in a supplier-manufacturer system. It is clear that this problem is harder than those corresponding problems studied in the earlier chapters. Therefore, we focus our study on the basic models, with a single supplier S, a single manufacturer M and a single customer C as depicted in Figure 7.1, where the supplier and the manufacturer are considered as a system.

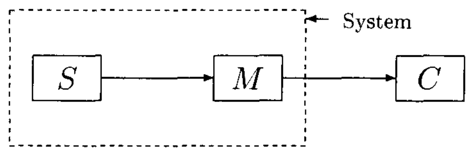


Figure 7.1: A Supplier-manufacturer system with a single customer.

There are N jobs to be processed at the supplier and delivered to the manufacturer; the manufacturer in turn processes the arrived jobs and delivers the finished jobs to the customer. Job J_i needs times $p_i^{(s)}$ and $p_i^{(m)}$ to process at the supplier and manufacturer, respectively. A holding cost of $w_i^{(s)}$ and $w_i^{(m)}$ occur for job J_i at the supplier and at the manufacturer, respectively. Also associated with each delivery batch from the supplier to the manufacturer is a delivery cost d_s and from the manufacturer to the customer is a delivery cost d_m. We have to find the optimal number of delivery batches $n^{(s)}$, batch sizes $b_i^{(s)}$ and the delivery times u_i (for $i = 1, 2, ..., n^{(s)}$) for the supplier, and the optimal number of delivery batches $n^{(m)}$, batch sizes $b_i^{(m)}$ and the delivery times v_i (for $i = 1, 2, ..., n^{(m)}$) for the manufacturer so that the total cost of the system, TC, is minimized.

$$\begin{aligned} TC &= \sum_{i=1}^{n^{(s)}} u_i \sum_{j=b_{i-1}^{(s)}+1}^{b_i^{(s)}} w_j^{(s)} + \sum_{i=1}^{n^{(m)}} v_i \sum_{j=b_{i-1}^{(m)}+1}^{b_i^{(m)}} w_j^{(m)} - \sum_{i=1}^{n^{(s)}} u_i \sum_{j=b_{i-1}^{(s)}+1}^{b_i^{(s)}} w_j^{(m)} + n^{(s)} d_s + n^{(m)} d_m \\ &= \sum_{i=1}^{n^{(s)}} u_i \sum_{j=b_{i-1}^{(s)}+1}^{b_i^{(s)}} (w_j^{(s)} - w_j^{(m)}) + n^{(s)} d_s + \sum_{i=1}^{n^{(m)}} v_i \sum_{j=b_{i-1}^{(m)}+1}^{b_i^{(m)}} w_j^{(m)} + n^{(m)} d_m \qquad (7.1) \end{aligned}$$

In Equation 7.1, the first and the second terms are associated with the supplier's delivery batches, and the third and the fourth terms are associated with the manufacturer's delivery batches. Based on this, we develop our batch scheduling algorithms.

7.2 Batch Scheduling of a Given Job Sequence

In this section, we develop dynamic programming algorithms to find the optimal batch scheduling in a supplier-manufacturer system, where the supplier and the manufacturer follow the same given job processing sequence. Without loss of generality, let the job processing sequence be $J_1, J_2, \ldots, J_N$.

Property 7.1 *There exists an optimal batch schedule in which a batch is delivered from the manufacturer to the customer as soon as the last item of that batch is completed processing.*

7.2.1 Dynamic Programming Algorithms

We develop dynamic programming algorithms to find the optimal batching schedule in a supplier-manufacturer system in which the supplier and the manufacturer process jobs in the same given job sequence. We analyse the problem in two cases:
(i) holding cost at the supplier is less than the holding cost at the manufacturer, i.e., $w_i^{(s)} < w_i^{(m)}$ for $(i = 1, 2 \ldots, N)$
(ii) holding cost at the supplier is greater or equal to the holding cost at the manufacturer, i.e., $w_i^{(s)} \geq w_i^{(m)}$ for $(i = 1, 2, \ldots, N)$.

The holding cost of a job is generally characterized by the warehouse costs, insurance, and interest rate in addition to the value of the job. Therefore it is reasonable to assume that holding costs at the manufacturer are larger(smaller) than the holding costs at the supplier for all the jobs.

Case 1 $w_i^{(s)} < w_i^{(m)}$ **for** $i = 1, 2, \ldots, N$

Lemma 7.1 *In the optimal schedule, a batch is delivered from the supplier to the manufacturer only when there is no item waiting for processing at the manufacturer.*

Proof: Consider an optimal schedule which contradicts the lemma. Then in that optimal schedule, there is at least one batch $b_k^{(s)}$ which is delivered to the manufacturer at time t_1 while the items waiting for processing at the manufacturer at t_1 are completed processing at t_2 $(t_2 > t_1)$. Now move the delivery time of $b_k^{(s)}$ to

t_2. Note that this shift in the delivery time does not affect processing and delivery schedules of other batches at the supplier and the manufacturer. However, this shift will increase the holding cost at the supplier by $(t_2 - t_1) \sum_{i=b_{k-1}^{(s)}+1}^{b_k^{(s)}} w_i^{(s)}$ and decrease the holding cost at the manufacturer by $(t_2 - t_1) \sum_{i=b_{k-1}^{(s)}}^{b_k^{(s)}} w_i^{(m)}$. Therefore the net change in the total cost of the supplier-manufacturer system due to this shift is $\Delta = (t_2 - t_1) \sum_{i=b_{k-1}^{(s)}+1}^{b_k^{(s)}} (w_i^{(s)} - w_i^{(m)}) < 0$. This contradicts the optimality assumption.
□

We develop the forward dynamic programming algorithm for Case 1 based on Lemma 7.1.

Dynamic Programming Algorithm for Case 1

Let $f(x, a, y, t_s)$ be the minimum total cost of the system when the supplier has delivered x items to the manufacturer with the last batch of size a delivered at time t_s, and the manufacturer has delivered $y \leq x$ items to the customer.

The state variables x, a, y and t_s change only when the supplier delivers a batch to the manufacturer or when the manufacturer delivers a batch to the customer. The supplier's delivery changes the state variables x, a and t_s, and the manufacturer's delivery changes the state variable y.

Let $T = \sum_{i=1}^{N} (p_i^{(s)} + p_i^{(m)})$.

Forward Recurrence Relation:

$$f(x,a,y,t_s) = \min \begin{cases} \min\limits_{a' \leq x-a, t'_s \in T} f(x-a, a', y, t'_s) + t_s \sum\limits_{i=x-a+1}^{x} \left(w_i^{(s)} - w_i^{(m)}\right) + d_s \\ \min\limits_{y'<y} f(x, a, y', t_s) + d_m + (t_s + \sum\limits_{i=(x-a)+1}^{y} p_i^{(m)}) \sum\limits_{i=y'+1}^{y} w_i^{(m)} \end{cases}$$

In the recurrence relation, the first term finds the minimum cost when the supplier delivers a batch of size a at time t_s, and the second term finds the minimum cost when the manufacturer delivers a batch of size $(y - y')$ to the customer.

Boundary Condition: $f(0,0,0,0) = 0$; $f(x,.,y,.) = \infty$ for all $y > x$; $f(x,a,.,.) = \infty$ for all $a > x$;

Optimal Solution Value: $\min_{a \leq N;\ \forall\ t_s} \{f(N,a,N,t_s\}$.

Theorem 7.1 *Algorithm 7.1 finds the optimal batching schedule in the supplier-manufacturer system for the case 1 in $O(N^4T^2)$ time, where $T = \sum_{i=1}^{N} \left(p_i^{(s)} + p_i^{(m)}\right)$.*

Proof: The algorithm looks at all feasible values for decision variables. Therefore, it finds the optimal solution. For a given x, a, y, t_s value there will be at most $O(NT)$ operations. Thus the total complexity is $O(N^4T^2)$. □

Case 2 $w_i^{(s)} \geq w_i^{(m)}$ for $i = 1, 2, \ldots, N$

Lemma 7.2 *There exists an optimal batch schedule in which a batch is delivered from the supplier to the manufacturer as soon as all the items belonging to that batch have completed processing at the supplier.*

Proof: Let us assume that the lemma does not hold. Then there will be at least one batch $b_k^{(s)}$ delivered from the supplier to the manufacturer at time t_2 while $b_k^{(s)}$ has already been completed at the supplier at t_1 ($t_1 < t_2$). Now move forward the delivery time of $b_k^{(s)}$ to t_1. This would result in decreased holding cost at the supplier by $(t_2 - t_1) \sum_{i=b_{k-1}^{(s)}+1}^{b_k^{(s)}} w_i^{(s)}$ and increased holding cost at the manufacturer by $(t_2 - t_1) \sum_{i=b_{k-1}^{(s)}+1}^{b_k^{(s)}} w_i^{(m)}$. Therefore the total change in the holding cost in the supplier-manufacturer system, $\Delta = (t_2 - t_1) \sum_{i=b_{k-1}^{(s)}+a}^{b_k^{(s)}} (w_i^{(m)} - w_i^{(s)}) \leq 0$. Further note that this shift does not affect the processing and delivery batch schedules of all other batches at the supplier and manufacturer. Thus this contradicts the optimality assumption.

□

From Lemma 7.2, we know that at the supplier the delivery time of any batch whose last item is the xth job is $\sum_{i=1}^{x} p_i^{(s)}$. Based on this property, we develop a dynamic programming algorithm to find the optimal batching of the system for the case $w_i^{(s)} \geq w_i^{(m)}$.

Algorithm 7.2: Dynamic Programming Algorithm for Case 2

Let $f(x, y, t_m)$ be the minimum total cost of the system when the supplier has delivered x items to the manufacturer, and the manufacturer has delivered y items to the customer with the last delivery batch from the manufacturer being delivered at time t_m.

Recurrence Relation:

$$f(x,y,t_m) = \min \begin{cases} \min\limits_{x'<x} f(x',y,t_m) + d_s + \sum\limits_{i=1}^{x} p_i^{(s)} \sum\limits_{i=x'+1}^{x} (w_i^{(s)} - w_i^{(m)}) \\ \\ \min\limits_{y'<y;\sum_{i=1}^{y'} p_i^{(m)} \leq t'_m \leq t_m - \sum_{i=y-y'+1}^{y} p_i^{(m)}} f(x,y',t'_m) + d_m + \sum\limits_{i=y'+1}^{y} w_i^{(m)} t_m \end{cases}$$

In the recurrence relation, the first term finds the minimum cost when the supplier delivers a batch of size $(x - x')$, and the second term finds the minimum cost when the manufacturer delivers a batch of size $(y - y')$ to the customer at time t_m.

Boundary Condition: $f(0,0,0) = 0$; $f(x,y,.) = \infty$, for all $y > x$.
Optimal Solution Value: $\min\limits_{\forall\ t_m} \{f(N, N, t_m\}$.

Theorem 7.2 *Algorithm 7.2 finds optimal batch scheduling in a supplier-manufacturer system in $O(N^8)$ time.*

Proof: This algorithm is similar to the one developed by Hall and Potts [19] for the combined problem to minimize sum of completion times. Hall and Potts prove that for a given job sequence, there are only a polynomial number of values for the set of all possible completion times at the manufacturer. They prove that the number of values for the completion times is in $O(N^{g+h-1})$, where g is the number of manufacturers and h is the number of customers in the supply chain. Therefore, for our problem, the number of possible values for t_m is in $O(N^3)$. There will be $O(N^2)$ possible combinations for (x, y) and for each (x, y, t_m) there will be $O(N^3)$ operations. Thus the complexity is $O(N^8)$. □

Remark 7.1: In a supplier-manufacturer system with multiple copies of single product manufacturing, job sequence does not matter. Thus the dynamic programming algorithms Algorithm 7.1 and Algorithm 7.2 can be used to find optimal batch schedule for a single product.

Chapter 8

Conclusions and Future Research

This thesis studied batch scheduling in a supply chain. Chapters 3 and 4 studied the optimal batch scheduling problem in a supply chain from the viewpoint of a single supplier who services demand for multiple products by multiple customers. The supplier's system was assumed to have a single stage and was modeled by a single machine with possible setups. We developed an efficient polynomial time algorithm for the single product batch scheduling problem at the supplier using the property that the total cost function is discrete convex in the number of batches. Then for the problems with identical processing times or identical weights we proved that job sequencing and batching are separable and we provide polynomial time algorithms using an algorithm for a special shortest path problem. The algorithms can easily be modified to handle problems with possible practical restrictions on the maximum number of batches or batch sizes and different trucks with different capacities and delivery costs. Batch scheduling problems with arbitrary processing times and arbitrary weights are $NP-$hard. We provided a 2-approximation algorithm for this problem with single customer. Further, we proved that the approximation ratio of the algorithm decreases with increasing delivery costs and it approaches to 1 when the delivery costs are very large. We also extended the results of the single customer model to the batch scheduling problem with multiple customers.

Chapter 5 focused on batch scheduling problems at the manufacturer in a supply chain. It is clear that batch scheduling problems at the manufacturer are harder than those at the supplier. We first analyzed some polynomially solvable problems of the manufacturer. Multiple product batch scheduling problems at the manufacturer are NP-hard problems even when jobs have identical weights. Therefore we developed a 2-approximation algorithm for the identical weight problem and a hybrid meta-heuristic algorithm for the problem with arbitrary processing times and arbitrary weights. We developed a lowerbound for the weighted case using unit processing time model and then compared the performance of the heuristic algorithm with our lower bound.

In Chapter 6, we studied batch arrival scheduling problems at the manufacturer in a customer-centric supply chain where promised job due dates are considered constraints which must be satisfied. We showed that the problems are closely related to batch scheduling problems on a single machine with flow-time related objectives. We proved that minimizing the sum of total weighted flow time and delivery costs is strongly NP-hard. For the unweighted version of the problem, we presented efficient solution algorithms both for single machine and assembly systems. We also developed a dynamic programming solution for minimizing the sum of maximum flow time and delivery costs.

Finally Chapter 7 studied coordination and batch scheduling in a supplier-manufacturer system. We analyzed a basic model with a single supplier, a single manufacturer, and a single customer. We developed a dynamic programming algorithm to solve the batch scheduling problem of given job sequence in the supplier-manufacturer system.

Future research in this area may look at alternative objective functions or look for efficient heuristic or approximating solutions for the computationally difficult weighted cases at the manufacturer and the supplier-manufacturer system. We were unable to provide an example to check the tightness of our approximation algorithm. In some problems we only consider single manufacturer and/or single end-customer. Adding more manufacturers and/or customers may be a challenging work in the future.

Bibliography

[1] S. Albers and P. Brucker. (1993) The complexity of one-machine batching problems. *Discrete Applied Mathematics* **47**, 87-107

[2] U. Bagchi and R.H. Ahmadi. (1986) An improved lower bound for minimizing weighted completin times with deadlines. *Operations Research* **35**, 311-313

[3] S.P. Bansal. (1980) Single machine scheduling to minimize weighted sum of completion times with secondary criterion- A branch and bound approach. *European Journal of Operational Rsearch* **5**, 177-181

[4] J.C. Bean. (1994) Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing* **6**, 154-160

[5] R. Bhatnagar, P. Chandra and S.K. Goyal. (1993) Models for multi-plant coordination. *European Journal of Operational Research* **67**, 141-160

[6] BMW. (2004) Custom cars on demand:The automaker uses a pull system to build a customer-specified bbehucles within 10 days of order placement. *Modern Materials Handling*

[7] R.N. Burns. (1976) Scheduling to minimize the weighted sum of completion times with secondary criteria. *Naval Research Logistics Quarterly* **23**, 125-129

[8] P. Chandra and M.L. Fisher. (1994) Coordination of production and distribution planning. *European Journal of Operational Research* **72**, 503-517

[9] Z.L. Chen. (1997) Scheduling with batch setup times and earliness-tardiness penalties. *European Journal of Operational Research* **96**, 518-537

[10] Z.L. Chen and N.G. Hall. (2000) Supply chain scheduling: Assembly systems. Working Paper

[11] R. Cheng, M. Gen and Y. Tsujimura. (1996) A tutorial survey of job shop scheduling problems using genetic algorithms - I: Representation. *Computers and Industrial Engineering* **30**, 983-997

[12] T.C.E. Cheng, V.S. Gordon and M.Y. Kovalyov. (1996) Single machine scheduling with batch deliveries. *European Journal of Operational Research* **94**, 277-283

[13] S. Chopra, and P. Meindl. (2004) Supply Chain Management: Strategy, Planning, and Operation. Second Edition, Pearson Prentice Hall.

[14] E.G. Coffman, Jr., M. Yannakakis, M.J. Magazine and C. Santos. (1990) Batch sizing and job sequencing on a single machine. *Annals of Operations Research* **26**, 135-147

[15] L. Davis. (1985) Job shop scheduling with genetic algorithms. Proceedings of 1st International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, pp. 136-140

[16] G. Dobson, U.S. Karmarkar and J.L. Rummel. (1987) Batching to minimize flow times on one machine. *Management Science* **33**,784-799

[17] H. Emmons. (1975) A note on a scheduling problem with dual criteria. *Naval Research Logistics Quarterly* **22**, 615-616

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

[19] N.G. Hall and C.N. Potts. (2003) Supply chain scheduling: Batching and Delivery. *Operations Research* **51**, 566-584

[20] H. Heck and S. Roberts. (1972) A note on the extension of a result on scheduling with secondary criteria. *Naval Research Logistics Quarterly* **19**, 403-405

[21] D.S. Hochbaum and R. Shamir. (1991) Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research* **39**, 648-653

[22] J.H. Holland. (1975) Adaptation in natural and artificial system, Ann Arbor, The University of Michigan Press

[23] H.L. Lee and C. Billington. (1992) Managing supply chain inventory: pitfalls and opportunities. *Sloan Management Review/Sprint*, 65-73

[24] C.Y. Lee and Z.L. Chen. (2001) Machine scheduling with transportation considerations. *Journal of Scheduling* **4**, 3-24

[25] J.K. Lenstra, A.H.G. Rinnoy Kan, and P. Brucker. (1977) Complexity of machine scheduling problems. *Annals of Discrete Mathematics* **1**, 343-362

[26] Y. Monden. Toyota production system, Second Edition, 1993.

[27] C.L. Monma and C.N. Potts. (1989) On the complexity of scheduling with batch setup times. *Operations Research* **37**, 798-804

[28] D. Naddef and C. Santos. (1988) One-pass batching algorithms for the one machine problem. *Discrete Applied Mathematics* **21**, 133-145

[29] Y. Pan. (2003) An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time. *Operations Research Letters* **31**, 492-496

[30] M.E. Posner. (1984) Minimizing weighted completion times with deadlines. *Operations research* **33**, 562-574

[31] C.N. Potts and M.Y. Kovalyov. (2000) Scheduling with batching: A review. *European Journal of Operational Research* **120**, 228-249

[32] C.N Potts and L.N. Van Wassenhove. (1992) Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of Operational Research Society* **43**, 395-406

[33] C.N Potts and L.N. Van Wassenhove. (1983) An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *Journal of Operational Research Society* **43**, 395-406

[34] C. Santos and M. Magazine. (1985) Batching in single operation manufacturing systems. *Operations Research Letters* **4**, 99-103

[35] David F. Shallcross. (1992) A Polynomial algorithm for a one machine batching problem. *Operations Research Letters* **11**, 213-218

[36] E. Selvarajah and G. Steiner. (2004) Batch scheduling in a two-level supply chain: A focus on the supplier. *European Journal of Operational Research*, to appear.

[37] W.E. Smith. (1956) Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3**, 59-66

[38] G. Steiner and P. Stephenson. (2004) Pareto optima for total weighted completion time and maximum lateness on a single machine. Working Paper

[39] D.J. Thomas and P.M. Griffin. (1996) Coordinated supply chain management. *European Journal of Operational Research* **94**, 1-15

[40] S. Treville, R. D. Shapiro, A. Hameri. (2004) From supply chain to demand chain: the role of lead time reduction in improving demand chain performance. *Journal of Operations Management* **21**, 613-627

[41] Webster, S., K.R. Baker. (1995) Scheduling groups of jobs on a single machine. *Operations Research* **43**, 692-703

[42] Williams, J.F. (1981) A hybrid algorithm for simultaneous scheduling of production and distribution in multi-echelon structures. *Management Science* **29**, 77-92

[43] X. Yang. (2000) Scheduling with generalized batch delivery dates and earliness penalties. *IIE Transactions* **32**, 735-741