Minimizing the sum of flow times with batching and delivery in a Supply Chain

A Thesis submitted for the degree of Doctor of Philosophy

by
Mohammad Mahdavi Mazdeh

School of Engineering and Design, Brunel University.

May 2005

Acknowledgement

I have to indicate my best thankfulness to Professor Khalil Hindi with whose supervision I started my PhD. Although he left the UK but, always felt responsible about my work and I used his ideas and comments throughout this work.

I also have to indicate my best thankfulness to Professor Mansoor Sarhadi, who kindly accepted the supervision of this project after Professor Hindi left. He supported and encouraged me in the all aspects of my work.

In addition, I would like to thank Professor Malcolm Irving, my second supervisor for his support and kindness.

I would finally like to appreciate Mr. Habib Nehzati who did a great deal of help in coding of the algorithms and was always there for me.

Abstract

The aim of this thesis is to study one of the classical scheduling objectives that is of minimizing the sum of flow times, in the context of a supply chain network. We consider the situation that a supplier schedules a set of jobs for delivery in batches to several manufacturers, who in turn have to schedule and deliver jobs in batches to several customers.

The individual problem from the viewpoint of supplier and manufacturers will be considered separately. The decision problem faced by the supplier is that of minimizing the sum of flow time and delivery cost of a set of jobs to be processed on a single machine for delivery in batches to manufacturers. The problem from the viewpoint of manufacturer is similar to the supplier's problem and the only difference is that the scheduling, batching and delivery decisions made by the supplier define a release date for each job, before which the manufacturer cannot start the processing of that job.

Also a combined problem in the light of cooperation between the supplier and manufacturer will be considered. The objective of the combined problem is to find the best scheduling, batching, and delivery decisions that benefit the entire system including the supplier and manufacturer.

Structural properties of each problem are investigated and used to devise a branch and bound solution scheme. Computational experience shows significant improvements over existing algorithms and also shows that cooperation between a supplier and a manufacturer reduces the total system cost of up to 12.35%, while theoretically the reduction of up to 20% can be achieved for special cases.

Table of contents

Introduction

Supply chain management (SCM) has attracted a great deal of interest in the last few decades. SCM is referred to as an *integrated system* which coordinates a series of inter-related business processes. The key concept that distinguishes it from its constitutive components is integration across the chain. Although the general literature on SCM is extensive, there is a lack of literature on supply chain scheduling models, i.e. the literature that consider the benefit of coordination between different stages of a SC network from the viewpoint of scheduling models.

The aim of this thesis is to study one of the classical scheduling objectives that is of minimizing the sum of flow times (completion times), in the context of a supply chain network. We consider the situation that a supplier schedules a set of jobs for delivery in batches to several manufacturers, who in turn have to schedule and deliver jobs in batches to several customers.

We first consider the problem from the viewpoint of supplier. Here the objective is to minimize the total flow times (completion times) plus delivery costs. This is a natural extension of the problem of minimizing total flow times to cater for coordination between scheduling and distribution in a supply chain network. The problem contains an additional term, which is the delivery cost for each manufacturer.

Then, we consider the problem from the viewpoint of manufacturer. The batching and delivery decisions made by the supplier define a release date for each job before which time the manufacturer cannot process any jobs. Therefore, from the viewpoint of manufacturer the objective adopted is that of minimizing the sum of flow times (completion times) and delivery costs in presence of release dates. Here, the additional term is the delivery cost for each customer.

Batching and delivery decisions made by the supplier may not be ideal from the viewpoint of manufacturer. He may prefer to receive some batches earlier to achieve better utilization of machines, or to receive some batches later for reducing inventory

costs. Then, manufacturer may suggest a new schedule for accepting the batches, but it is not guaranteed that the supplier accepts the manufacturer's suggestion. The supplier may refuse the suggested schedule due to the resulting high delivery or inventory cost. This is a controversial issue, which is better handled through solving a combined problem that leads to the benefit of both parties in light of cooperation. Therefore, a combined problem in the context of cooperation between the supplier and manufacturer is also considered. The gain of the combined problem is to find the best scheduling, batching, and delivery decisions that benefit the entire system including the supplier and manufacturer.

Structural properties of each problem are investigated and used to devise a branch and bound solution scheme. Computational experience shows significant improvements over existing algorithms and also shows that cooperation between a supplier and a manufacturer reduces the total system cost of up to 12.35%, while theoretically the reduction of up to 20% can be achieved for special cases.

The outline of this thesis is as follows:

Chapter 1: This chapter presents a picture of the supply chain management and mathematical models in SCM, and briefly surveys the related literature from SCM perspective.

Chapter 2: Scheduling, its definition, framework, notation and standard problem forms will be reviewed in this chapter. Furthermore, dynamic programming and branch and bound methods are discussed and the literature relevant to the problems considered in this thesis are surveyed.

Chapter 3: In this chapter, the problem from the viewpoint of supplier, which is the problem of scheduling a set of jobs to be processed on a single machine by supplier for delivery in batches to manufacturers, will be considered.

Chapter 4: This chapter considers the problem from the viewpoint of manufacturer. This problem is similar to the supplier's problem and the only difference stems from the fact that batching and delivery decisions made by the supplier define a release date for each job.

Chapter 5: A combined problem in the light of cooperation between the supplier and manufacturer will be considered in this chapter.

Chapter 6: This brief chapter compares the results of total flow times plus delivery cost over the system, before and after cooperation between the supplier and manufacturer.

Chapter 7: Conclusions and further works will be presented in this chapter.

Chapter 1

1. Supply Chain Management

Supply chain management (SCM) has attracted a great deal of interest in last decades. Many businesses, manufacturers and suppliers seek to organise their work in accordance with the principles of SCM. Considerable research works have been focused in various aspects of SCM and it is likely that this trend will continue in the foreseeable future. It is noted that SCM is "at the cutting edge of reengineering" [50]. What exactly is SCM then and why it is so important? This is a question we try to survey briefly in section 1.1, where we consider some definitions and important key words of SCM. In section 1.2, we will focus on mathematical modelling in SCM and finally in section 1.3, a literature survey related to the subject of this dissertation will be offered.

1.1 Introduction and definitions

Traditionally, the various stages of supply chain such as procurement, production and distribution have been organized independently. Furthermore, the different functions of an organisation, including assembly, storage, and dispatching of finished product have traditionally focused their efforts on making effective decision within their own facilities. The reasons for this treatment stemmed from the facts that:

1-The objectives of different facilities or organizations are most often in conflict. For example, marketing objective of achieving maximum sales and high customer service can be in conflict with manufacturing objectives. Also, manufacturing

objectives can be in conflict with distribution centre objectives because many manufacturing operations are designed to derive maximum benefit through maximizing throughput and lower costs while the impact of their output on inventory levels and distribution centre may not be considered fully [42].

2- The complexity of making decision is reduced when each component is treated independently [87]. It is obvious that developing models for multi-echelon systems is more complex than single-echelons (see section 1-3).

Despite its simplicity, this treatment can have costly results. The costs of poor coordination, especially in global marketing, can be extremely high. Using the traditional way, companies require inventory at various locations throughout the chains while the costs of maintaining inventory are very high. In addition, the cost of holding inventory at any locations also means that most companies cannot provide a low cost product when their funds are tied up in inventory. More importantly, it should be noted that producing a quality product is not enough. "Getting the products to customers when, where, how, and in the quantity that they want, in a cost-effective manner, constituted an entirely new type of challenge" [51].

However, in today's marketplace, managers have realized that actions taken by one member of the chain influence profitability of the all others in the chain [59]. The most successful manufacturers seem to be those that have made a tough linkage between their internal processes and external suppliers (upstream) and customers (downstream). Firms have found that it is no longer enough to manage their own organization; they must also be involved in the management of the network of all upstream firms that provide inputs (directly or indirectly), as well as the network of downstream firms responsible for delivery and after–market services of the product to the end customer [51]. Hence, if the supply chain is defined as a network that begins from supplier and finishes with the customer and includes the flow of raw materials to firms, the processes of production, warehousing and distribution, afterwards there is a need to a system that tries to tie these activities together on the basis of good relationships among the parties concerned. The term Supply Chain Management (SCM) which was originally introduced by consultants in the early 1980s [17] is a response to this need.

1.1.1 Definition

Unfortunately, the term SCM has been used for different purposes in the literatures and there is not an explicit definition that covers all applications of supply chain management [74] and [86]. The APICS Dictionary has described the supply chain as:

- *The processes from the initial raw materials to the ultimate consumption of the finished product linking across supplier-user companies; and*

- *The function within and outside a company that enable the value chain to make products and provide services to the customer* [68].

The Supply Chain Council describes the supply chain as:

The supply chain, a term increasingly used by logistic professionals, encompasses every effort involved in producing and delivering a final product from the supplier's supplier to the customer's customer. For basic processes-plan, source, make, deliver- broadly define these efforts, which include managing supply and demand, sourcing raw materials and parts, manufacturing and assembly, warehousing and inventory tracking, order entry and order management, distribution across all channels, and delivery to customers[68].

For a review to the different definitions and activities of SCM see [29], [32], [86] and [17]. Among these different definitions we select the definition reported by Lambert and Cooper [63] which is based on the definition suggested by Global Supply Chain Forum (GSCF). The definition of SCM as developed and used by them is as follows:

Supply Chain Management is the integration of key business processes from end user through original suppliers that provides products, services and information that add value for customers and other stakeholders.

In this definition, a supply chain management is referred to as an *integrated system*, which coordinates a series of inter-related business processes. These processes make a network which begins from the extraction of raw material from the earth, transporting of raw materials to firms, transforming these materials into

finished products, warehousing these products and finally distributing these products to retailers or customers. In addition, information exchange among various businesses (e.g. suppliers, manufacturers, stock clerks, distributors, retailers) is also one part of these processes. Figure 1 shows the procedure of these processes in different chains.

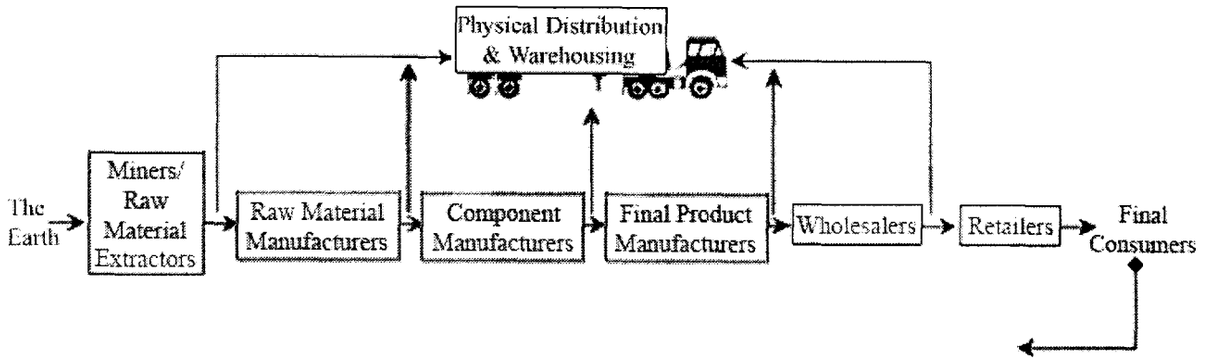


Fig 1-1: Activities and firms in a supply chain. Source: New and Payne [75].

1.1.2 Internal and external Supply Chain

Better understanding of supply chain management requires us to distinguish between internal and external supply chains.

The internal supply chain is that portion of a given supply chain that occurs within an individual organization, whereas the external supply chain extends to the key suppliers and customers [51]. In fact, the internal supply chain is a structure that could be seen in many businesses and it is not uncommon for an organization to have multiple links that span in different locations and different responsibilities. However, the external SC is more complex; a business organization has typically linkages to a wide range of suppliers, who supply a number of products and services and purchase materials for it, and also to multiple customers. Treatment and managing of these different organizations together in a whole SC on the basis of good relationships among the parties concerned is not easy. There are several issues that should be noticed for selecting external members. First, the competitive situation that exists between prospective SCM members has to be identified. The task of enhancing cooperation between two companies that are members of one supply chain, while being competitors in other markets, seems difficult if not impossible. In other words

the SCM organization will be more productive when the participants are not direct competitors in other markets. Secondly, all companies and their representatives that wish to participate in a SCM must follow similar goals. It is worth noting that 'similar goals' do not mean that members should have identical goals, but "their respective goals must be compatible with the overall SCM initiative". Thirdly, SCM have a great potential when all organisations involved with the chain feel their involvement is beneficial to their own business. In an internal supply chain one part of organisation may not benefit from a process but still cooperate to follow the goals of the organisation, while in an external supply chain such cooperation is meaningless [51].

1.1.3 Global supply chain

Astonishing developments in the computer and information technology has led to the development of the global marketplace. It is believed that "the emergence of global marketplace necessitates that SCM must be refocused in a global network context. In this context, the term used to capture the integration of activities and processes among organizational entities is referred to as global supply chain management (GSCM)" [53]. Clearly the aim of SCM and GSCM is the same with the only difference stemming from their context.

More concisely a global supply chain (GSC) is an extension of the external supply chain. Global supply chain occurs when some of the external portions of the supply chain (key supplier and/or customers) are abroad and the business has linkages to them. It is worth noting that there is difference between an internal supply chain within an individual organisation that has some links abroad and a GSC that links several organisations [51]. However, the GSCM are more difficult and complex than domestic SCM. Different taxes and duties, differential exchange rates, trade barriers, transfer prices, sources of uncertainty such as government stability and general infrastructure of particular country, are critical issues that should be considered when designing any global supply chain [89].

1.2 Supply chain decisions area

There are two approaches in the literature for classifying supply chain decision area: (1) classification based on temporal consideration and, (2) classification based on functional consideration.

1.2.1 Classification based on temporal consideration

Generally, SCM decisions based on temporal consideration can be classified into 2 broad categories: strategic and operational. Strategic decisions are concerned with how to link the essential parts that affect the process of SCM, while operational decisions focus on one part of the process. Strategic decisions are long term; normally more than one year, and make SCM policies, whereas operational decisions are short term and active day to day. Strategic decisions are global and try to incorporate all aspects of SCM, while operational decisions are local and organise each part [90].

Accepting this classification, Ganeshan and Harrison [42] believe that there are four major decision areas in supply chain management including location, production, inventory, and transportation (distribution). They note that there are both strategic and operational elements in each of these decision areas.

Thomas and Griffin [87] define three categories of operational coordination that are: buyer-vendor, production- distribution, and inventory-distribution. For the strategic issues, they consider that it may include: "plant or distribution centre openings and closings, allocation of equipment to manufacturing facilities, selection of a location or locations for manufacturer of raw product, and evaluation of changes in the flow of particular product through the supply chain."

Another possible classification differentiates among strategic, tactical, and operational decisions [9]. Tactical planning usually handles more variables than operational planning and less than strategic planning and extends over 3 to 12 month. According to Min and Zhou [71]"the classes of supply chain problems encountered in strategic analysis include location-allocation decisions, demand planning, distribution channel planning, strategic alliances, new product development, outsourcing, supplier selection, information technology (IT) selection, pricing, and network restructuring." They emphasise that although most supply chain issues are strategic in nature, there are also some tactical problems. These issues include

"inventory control, production/distribution coordination, order/freight consolidation, material handling, equipment selection, and layout design". And finally the problems encountered with operational decisions include "vehicle routing/scheduling, workforce scheduling, record keeping, and packaging."

1.2.2 Classifying based on functional consideration

Johnson and Pyke [59], without clearly distinguishing between strategic and operational decisions, have divided SCM into twelve areas: location, transportation and logistics, inventory and forecasting, marketing and channel restructuring, sourcing and supplier management, information and electronic mediated environments, product design and new product introduction, service and after-sales support, reverse logistics and green issues, outsourcing and strategic alliances, metrics and incentives, and global issues. For each of these twelve areas, they have provided a brief description of the basic content and referred the reader to some published articles.

Biswas and Narhari [14] have divided supply chain into four major decision areas: procurement, manufacturing, distribution, and logistics. They suggest that there are strategic, tactical, and operational questions in each of these areas and refer to Shapiro [83] for details.

1.3 Supply Chain Modelling-mathematical approach

Although the literature dealing with decision models in SCM are extensive, only a few articles have tried to create a topology or taxonomy of mathematical models. Traditionally, decision variables of different stages in supply chain have been optimized separately and in this way the complexity of models were reduced. The idea of coordinated decision between different stages began about 1960 by Clark and Scarf when they studied the multi-echelon inventory/distribution systems [87]. Since that time many researches have investigated the multi-echelon inventory and distribution systems. Some of these notable efforts have been addressed in sections 1.4 and 2.6. It is obvious that developing models for multi-echelon systems is more complex than a single-echelon. Some of the most important reasons for this complexity according to Biswas and Narhari [14] are: "large scale nature of the

supply chain networks, hierarchical structure of decisions, randomness of various inputs and operations, dynamic nature of interactions among supply chain elements."

However, the key concept in SCM that discriminates it from the traditional activities (e. g. marketing, manufacturing, etc.) is the integration between different chains, horizontally or vertically. Significant efforts have to be devoted to creating such models. Since the supply chain management is a widespread and interdisciplinary subject, no model can cover all aspects of supply chains. Clearly, each of the strategic, tactical and operational categories requires a different model. Models, which are able to describe strategic decisions, are huge and need a considerable amount of data and therefore these models often provide approximate solution to the decisions they describe. On the other hand, operational decisions are short term, and focus on activities on a day-to-day basis. Therefore these models due to their perspective often consider great details and provide optimal or near optimal solutions for the operational decisions [42].

However, model performances vary from chain to chain and from business to business. As a result, numerous mathematical models have been developed to cater for the various problems encountered.

1.3.1 Taxonomy of Supply Chain Modelling-mathematical approach

In this section we try to give a taxonomy of mathematical models that can be applied to performing and analysing SCM problems. It should be noticed that this taxonomy is different from Operational Research (OR) models on SCM. For a review on OR/SCM models, the reader is referred to Shapiro [82], Arns et al. [7], Huan et al. [56].

Generally, mathematical models can be divided into two broad categories: deterministic and stochastic [10] and [54]. In deterministic models, the parameters - are known exactly; while in the stochastic models there is at least one parameter that is assumed to follow a particular probability distribution.

According to Beamon [11] the supply chain design and analysis can be divided into four categories of mathematical modelling approach. These four categories are: deterministic analytical models, stochastic analytical models, economic models, and simulation models.

Shapiro [83] and [82] has highlighted the role of information technology (IT) for the purpose of integrated supply chain. He has distinguished between transactional IT and analytical IT and then supposed that analytical IT involves the implementation and application of two types of mathematical models: *descriptive models* and *normative models*. *Descriptive models* are models that should be developed to clear functional relationships in each chain and outside the chain. These models include forecasting models, cost relationships, resource utilization relationships and simulation models. N*ormative models* are models that help the manager to make better decisions, while the term '*normative*' refers to processes for identifying norms that the company should try to achieve. He supposes that *normative models* and *optimization models* are synonyms for *mathematical programming models,* while *mathematical programming models* include linear programming, mixed integer programming, non linear programming, stochastic programming and all other classes and methods that have been studied in field of operations research. He remarked that "descriptive models are necessary but not sufficient for realizing effective decision making and similarly, an accurate management accounting model of manufacturing process costs is necessary but not sufficient to identify an optimal production schedule"[82].

Min and Zhou [71] by referring to Budnick et al. [15] , Silver [84] and Zipkin [96] noted that "some supply chain models based on inventory theory and simulation contain both deterministic and stochastic elements and consequently should be treated as hybrids". They also suggest that another category called 'IT-driven models' has to be added to the taxonomy to reflect the current advances in IT for improving supply chain efficiency. Therefore, by their taxonomy the supply chain models are divided into four major categories: deterministic (non-probabilistic), stochastic (probabilistic), hybrid, and IT-driven.

1.4 Literature Survey

Although the general literature on SCM is extensive, there is a lack of literature on supply chain scheduling models, i.e. the literature that consider the benefit of coordination between different stages of a SC network from the viewpoint of scheduling models. There appears to exist only one paper that has explicitly

addressed the type of problem, being considered in this thesis. We will consider this paper and also some papers with similar spirit to our work will be considered.

Vidal and Goetschalckx [89] offer an extensive survey of strategic production-distribution models on global supply chain. Their special attention is focused on considering the efficiency of mixed integer programming (MIP) models for solving these kinds of problems. They conclude that although there is not a consensus on the efficiency of MIP, the important role of MIP models for considering global supply chain process cannot be relinquished. Moreover, they mention that the factor of uncertainty is not considered in the most formulation and also some international factors such as exchange rates, taxes and duties are not perfectly described by the existing models.

Weng [93] considers the situation in which one manufacturer and one distributor mutually share all relevant information to make the decision policies that maximize their individual expected profits. They operate to meet random demand for a product, which has a short product life cycle. He shows that neglecting coordination, especially when random demand is very sensitive to distributor's sales prices, or when the manufacturer's unit sales price is much higher than the manufacturer's unit cost, can be very costly. Weng [94] has extended his work by analyzing the same problem in confronts of risk. In this part, he shows that how by decreasing the risk, the expected benefit decrease and vice versa.

Sarmiento and Nagi [81] offer a review on integrated production-distribution systems. Based on previous studies they point out some substantial benefits of production-distribution integration with commenting that many aspects of such integration are not yet considered.

Erenguc, Simpson and Vakharia [41] survey the similar problem with some different formats. They emphasise on the role of information technology for further research and point out that since from individual chain perspective, sharing the information might reduce the competitive efficiency of company, then there is a need to investigate different aspects of such sharing information and provide a mechanism for it. They also mention the need of simulation models that optimize three stages of a supply chain network, i.e. supplier, distributor and manufacturer, simultaneously.

Croom, Romano and Giannakis [32] present an analytical survey with the aim of giving a framework for classification and analysis of major issues on SCM and also describing and evaluating the methodologies that are already used in SCM literature.

Their literature review shows that 83% percent of literature is empirical while only 17% of them are theoretical. In the empirical literature 27% is prescriptive and 56% is descriptive, while in the theoretical literature these percentages are 6% and 11% respectively. Accordingly they suggest the need for more research in the theoretical aspects of SCM.

Krajewski and Wei [62] have investigated the value of production schedule integration in supply chain. They have provided a model for evaluating the effect of some factors such as holding costs, supplier lead times, forecasting effectiveness, and schedule change costs on the value of sharing an integrated production schedule in a supply chain. Their study is based on performing the model on two supply chains involving 10 plants. One of their significant results is that schedule integration is not always beneficial and for some environments it is better to perform production schedule independently. These situations are discussed in the paper in detail.

Lee and Kim [65] in response to Erenguc et al. [41] provide a hybrid model in which the analytical and simulation models are combined for considering the problem of production-distribution. They hold the view that disregarding operation times, which is normally assumed in analytical models, decreases the reality of problem. To solve this problem they develop a simulation model which has a general production-distribution characteristic. Therefore, the machine operation time and distribution operation time constraints can be considered as stochastic factors by that model and the results will be adjusted by analytical model which is developed independently. Based on experimental result, they report that adjusted operation time of the system can have a significant impact on the production-distribution plan.

Chen and Paulraj [17]offered an extensive survey of over 400 articles on SCM to identify the constructs and measurement for supply chain management.

Elmahi et al. [40] consider the problem of batch delivery optimization in a supply chain under the just-in-time condition. They provide a genetic algorithm that minimizes the global advance time of the whole demand and moreover, maintains a minimal level of product in process. Based on experimental result, their algorithm provides the optimal solution in more than 80% of cases.

The explicit problems we will explore in this dissertation have been investigated by Hall and Potts [49]. Hall and Potts consider a variety of scheduling, batching and delivery problems that arise in an arborescent supply chain, where a supplier makes deliveries to several manufacturers, who also make deliveries to customers. The

objective is to minimize the overall scheduling and delivery costs, using several classical scheduling objectives. For each problem, they either derive an efficient dynamic programming algorithm (DP), or demonstrate that it is intractable. They demonstrate that cooperation between manufacturer and supplier can reduce the total system cost by at least 20% or 25% or even by up to 100%, depending on the scheduling objective. The details of each problem, appropriate to the problem that we will explore in this thesis, are explained in the next chapter.

1.5 The aim of this thesis

According to our best knowledge, the only paper that considers the benefit of cooperation between different stages of a supply chain network from the viewpoint of scheduling models is that of Hall and Potts [49]. They have highlighted it too that "the benefit and challenges of coordinated decision making within supply chain scheduling models have not been studied" before. Therefore, the most important feature of their work is that they have introduced a new class of research in the filed of supply chain management. They have provided a DP algorithm for each problem under consideration. Dynamic programming algorithm, as will be explained in the next chapter, takes advantage of overlapping sub-problems by solving each of them once, however it cannot handle large-scale problems. Although, the authors have not coded and tested their algorithms, it can be clearly observed that the complexity of their algorithms are such that by increasing the number of jobs manufacturers and customers the computing running time increases.

In this thesis we attempts to solve the problem of minimizing the sum of flow times with batching and delivery in different steps of a supply chain that includes a supplier, several manufacturers and several customers. Our approach for solving the problems is to provide a branch and bound methods rather than DP algorithms of Hall and Potts. Furthermore, the DP algorithms of Hall and Potts for the same problems are coded. The significant advantage of branch and bound algorithms over DP will be considered. It will be also discussed that for a combined problem, DP can only handle the problem instances with only a few customers and very restricted number of jobs. Since the main idea of coordination between different stages of supply chain can be found in the combined problem, improving the algorithms that

can handle the problems with more combination of manufacturers, customers and jobs is very essential.

Chapter 2

2. Scheduling

2.1 Definition

"Scheduling concerns the allocation of limited resources to tasks, over time. It is a decision making process that has the goal of optimizing one or more objectives" [78]. Scheduling problems are very common. They exist whenever there is a choice of selecting a number of tasks. The problem could be one of sequencing jobs in a machine shop, aircraft waiting for landing, programs to be run at a computing centre or usual tasks faced every day. Resources may be machines in workshop, runways at an airport, and so on. Each task could have a different priority, restriction for starting time, and due date. The objective function in a scheduling problem may take many forms, such as minimization of the completion time of the last job, or minimization of the number of tardy jobs and so on.

Scheduling is a decision making process that arises in most manufacturing and production systems, as well as in transportation and distribution settings. The scheduling functions have to interface with many other functions. These interfaces differ from one situation to another and also depend on system organization. For instance, the scheduling function is affected by the production planning process, which handles medium to long term planning for the entire organization. This latter process must consider inventory levels, forecasts, and resource requirements to optimize at a higher level the product mix and long term resource allocation. Clearly, decisions made by the planning function have an impact on scheduling.

2.2 Scheduling Models: mathematical approach

Sequencing and scheduling theory is primarily concerned with the development of mathematical models and techniques. Generally, mathematically based scheduling models are divided into two categories: deterministic models, and stochastic models. Deterministic models are models in which job data such as processing times, release dates and due dates are known and it is assumed that there are a finite number of jobs to be scheduled and a single objective to be minimized. In stochastic models, there are also a finite number of jobs to be scheduled, but jobs data are given in terms of probability distributions. In these models, a single objective has to be minimized.

There are a number of mathematical approaches for solving scheduling problems. Linear and nonlinear programming, integer and mixed integer programming, network analysis, dynamic programming, branch and bound method, game theory, etc. are some common mathematical tools that can be applied to solve a scheduling problem. Choosing a best method for solving a problem, which may be solvable in different ways, is a challenge that needs experience and insight. This challenge becomes more important, especially when it is proved that a problem cannot be solved in a polynomial time (see section 2.5). Appropriate with the materials of this thesis, in the next two subsections we introduce two important methods, which are dynamic programming and branch and bound methods.

2.2.1 Dynamic Programming

Dynamic Programming (DP) is a mathematical theory or a technique that was developed further by Bellman for solving multistage decision problems first time in 1950s. Since then, DP algorithm has had various applications in the areas of engineering, economics, commerce, management, etc.

What distinguishes dynamic programming from other optimization techniques, and particularly linear programming, is that although it cannot handle large scale problems, it is suitable for solving problems with more complexity than linear programming when the size is not large [24]. This method is usually used in optimization problems in which a set of decisions must be taken for determining an optimal solution. In the process of making decision, sub-problems of the same form often appear. The solution of each of the sub-problems will be stored. The efficiency

of DP stems from the fact that given sub-problems may arise more than once. Since the given sub-problem is solved and its solution is stored, the need for repeated calculation is therefore removed. This procedure reduces the amount of computation required [30].

Dynamic programming algorithm usually consists of 4 steps:

1. Characterize the structure of an optimal solution.
2. Recursively defines the value of an optimal solution.
3. Compute the value of an optimal solution in bottom-up fashion.
4. Construct an optimal solution from computed information.

2.2.1.1. Evaluating a problem

The following two observations are helpful in determining if a problem can be solved using dynamic programming:

1. Optimal substructure

If an optimal solution to a problem contains optimal solutions to sub-problems, we say that this problem exhibits optimal substructure. Whenever a problem exhibits optimal substructure, it is a good clue that dynamic programming might apply [30].

2. Overlapping sub-problems

When we develop a recursive algorithm to solve the problem, instead of always generating new sub-problems, the same sub-problems may arise over and over. We call these sub-problems "overlapping sub-problems". Dynamic programming algorithm takes advantage of overlapping sub-problems by solving each of them once and then storing the solution in a table where it can be looked up when needed. However, the space of sub-problem must be relatively "small" so that dynamic programming is applicable [30].

2.2.1.2 Basic terms and variables

Now let's introduce the basic terms and variables of dynamic programming algorithm:

1. Stage

The problem may be divided into a set of stages, with each stage requiring a policy decision. Any dynamic programming problem requires a set of interrelated decisions, where each decision corresponds to one stage of the problem.

2. State

Each stage contains a number of states. The states are the various possible situations at any given stage. The value assigned to each state can be interpreted as the intermediate contribution to the objective function. These values show the cost of transferring from one state of given stage to next stage and have to be stored somewhere for all states of each stage. In most problems, the objective is finding the minimum or maximum value through the problem. Then, the minimum or maximum value for transferring the problem from the current state of given stage to the next stage is crucial.

3. Decision

As cited above at a given state of each stage, there are different options to determine the next stage. Hence, a decision must be made for determining the best options. This decision is independent of the previous stage, i.e. it is not important how we have got to the current state. "For dynamic programming in general, knowledge of the current state of the system conveys all the information about its previous behaviour necessary for determining the optimal policy henceforth. This property is "*Marcovian property*" [54]".

4. Recursive relationship

The same decision logic applies to any given state in all stages. This gives this opportunity to create a recursive relationship that given the information of previous stages provides the optimal policy for the current stage.

2.2.1.3 Formulation

Formulating a dynamic programming problem may differ from case to case. However the essential idea is the same and the problem can be formulated as follows according to Hillier and Liberman [54]. Let:

N = Number of stages.

n = Label of current stage ($(n = 1, 2, ..., N)$.

s_n = Current state for stage n .

x_n = Decision variable for stage n .

x_n^* = Optimal value of x_n (given s_n).

$f(s_n, x_n)$ = Contribution of stages $n, n+1, ..., N$ to objective function if system starts in state s_n at stage n, immediate decision is x_n, and optimal decisions are made thereafter.

$f_n^*(s_n) = f_n(s_n, x_n^*)$.

The recursive relationship will always be of the form

$f_n^*(s_n) = \max_{x_n}\{f_n(s_n, x_n)\}$ or $f_n^*(s_n) = \min_{x_n}\{f_n(s_n, x_n)\}$,

where $f(s_n, x_n)$ would be written in terms of s_n, x_n, $f_{n\pm1}^*(s_{n\pm1})$, and probably some measure of the immediate contribution of x_n to the objective function.

Writing $f_n^*(s_n)$ in term of $f_{n\pm1}^*(s_{n\pm1})$ implies that they make a recursive relationship.

2.2.1.4 Forward and Backward DP

A choice can be made between forward and backward dynamic programming corresponding to how we approach the solution to the problem.

There is a forward dynamic programming method when the solution procedure starts from the first stage (n = 1) and moves forward stage by stage. In this case $f(s_n, x_n)$ would be written in terms of $f_{n-1}^*(s_{n-1})$ (other elements don't change). In contrast, it is a backward dynamic programming when the solution procedure starts at the end ($n = N$) and moves backward stage by stage. In this case $f(s_n, x_n)$ would be written in terms of $f_{n+1}^*(s_{n+1})$ (other elements don't change).

2.2.1.5 Deterministic and Stochastic DP

A Dynamic programming problem is deterministic when at each state of the problem, the next state of the problem can be completely determined by policy decision at the current stage. In contrast, the probabilistic or stochastic dynamic programming happens where there is a probability distribution for what the next state will be.

2.2.2 Branch and Bound method

Branch and Bound (B&B) method belongs to the class of implicit enumeration methods that was first proposed by Land and Diog in 1960 for linear programming

[1]. B&B algorithm is one of the efficient tools and exact methods that can be applied for solving various problems, especially $NP-$hard (see section 2.5) discrete optimization problems. The main idea in B&B stems from the fact that the entire enumeration search for a problem is impossible due to exponentially increasing number of potential solution space. Hence, the use of bound for the function to be optimized enables B&B method to search some parts of the solution space implicitly [26]. One of the most powerful aspects of B&B can be viewed in performance of Mixed Integer Programming. The details of B&B method can be found in most mathematical optimisation books, for example see Minux [72]. The general idea and terminology of this method will be described in the following section.

2.2.2.1 General idea and terminology

Assume K_0 is a set of feasible solutions for minimizing problem $f(K)$. The general idea of B&B can be compressed in two concepts; *Branching* and *Bounding*.

1. Branching

First concept is that during B&B, set K_0 can be partitioned into some simpler subsets while, each subset includes a set of feasible solutions. This is called *Branching*. The strategy chosen for solving a problem, and also given the type of the problem, the number of branches derived from the original problem might change. However, since the procedure will be repeated recursively, i.e. each subset of the original problem will be divided into new subset and so on, then the origin problem and all its subsets form a tree structure, called *search tree*.

Based on tree data structure terminology, the original problem is placed at *root node*. A root node is a specific node in a tree structure from which all operations start.

Each node will have many *children*, which will be generated in accordance to the branching strategy. A node with children is called *parent node*, then root node is a node that has some children but has no parent. A node with no children is called *leaf node*. Leaf nodes are the farthest from the root node while all variables at leaf nodes are fixed. A *child node* or *descendant node* is a node that is linked to by a parent node, i.e. node B is a child of node A if and only if node B is a successor node of A.

2. Bounding

Another concept of B&B is *bounding*, which is a way for *pruning* some parts of solution space without searching. This way is completely interrelated to the structure

of problem and is different form situation to situation. In a minimization problem such as $f(K)$, it is not usually difficult to find an initial value that is greater than or equal to every feasible solutions of set K_0. This value is called initial *upper bound.* Whenever a new node is generated, using the strategy, which is tightly dependent on the structure of problem, a *lower bound* will be calculated. *Lower bound* is defined as opposite of upper bound and it is the minimum value that $f(K)$ may have at each considering point of the problem. Calculating lower bound follows the strategy that one chooses for solving the problem and will be implemented in a different way for different problems. The core idea in B&B is that for each current node, if lower bound is grater than or equal to upper bound, this node will be *pruned.* It is worth noting that when a node is pruned, i.e. discarded, all subsets (children) that could be derived from this node will be discarded. This is the most important property of B&B method that allows decreasing the number of enumeration. Hence, providing good upper bound and lower bound is crucial and the merits of B&B method are entirely related to proper choices of upper and lower bounds.

A node is *fathomed* if:

1. It is a leaf node, i.e. all variables are fixed.
2. The lower bound exceeds or equals the upper bound.

In the process, the lower bound strategy may actually produce a minimum value at leaf node. This value has to be compared with the upper bound; if it is less than upper bound then the value of upper bound must be replaced by it. This value is called *incumbent* upper bound.

2.2.2.1 Breadth-first search and Depth-first search methods

There are several schemes for traversing or searching a tree. *Breadth-first search* and *depth-first search* are two of most important schemes that will be described here:

Breadth-first search is an uninformed search scheme that considers all nodes of a tree systematically. In this method we consider systematically all nodes on a given level of the tree before going deeper. For the following graph:

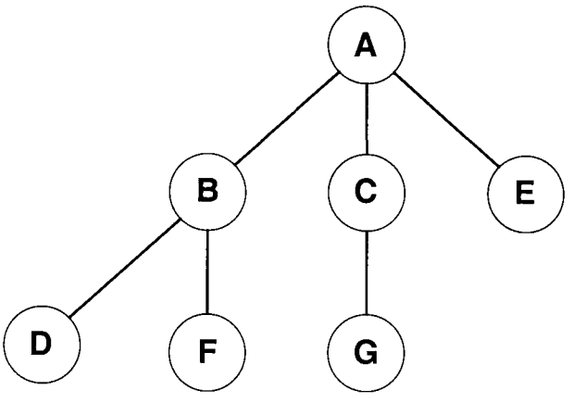


Figure 2-1.

a breadth-first search starting at A, and assuming that at each level we start from the node which is placed at the left side, will result in the following order: A, B, C, E, D, F, G.

In contrast to the breadth-first search, the depth-first search is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal state is found, or it hits a node that has no children. Then the search backtracks and starts off on the next node. Hence, in depth-first search method we go deep before going wide. We start at the root node and go as far as possible along a branch before backtracking. For the above graph a depthe-first search starting at A, if assuming that at each level we start from the node which is placed at the left side and also assuming that the search remembers previously-visited nodes and will not repeat them, will result in the following order: A, B, D, F, E, C, G [1].

2.3 Framework and notation

In this section we introduce some fairly standard notations that are used throughout this thesis. Let there be a finite number of jobs, n , and a finite number of machines, m . Subscript i refers to a machine and subscript j refers to a job. Normally, the pair (i, j) refers to the operation of job j on machine i .

Processing time ($p_{i,j}$). The $p_{i,j}$ represents the processing time of job j on machine i . The index i can be omitted if the processing time of job j is independent of the machine or when the job is processed only on one given machine.

Release date ***(r_j).*** The release date, or *ready date* or *ready time* r_j, is the earliest time that job j can start its processing and is the time that the job arrives at the system.

Due date ***(d_j).*** The due date is the time that is promised to customer for the processing of the job to be completed. If a job is completed after its due date, a penalty is incurred.

Weight ***(w_j).*** This is a priority factor, which refers to job j to show its importance or its cost relative to the other jobs in the system.

Completion time ***($C_{i,j}$).*** The completion time of job j on machine i is the time at which the processing of job j on machine i is finished. Similarly, the completion time of job j in a system is the time at which job j can exit the system and is denoted by C_j.

Flow time ***($F_{i,j}$).*** The flow time of job j is the amount of time job j spends on machine i. Similarly, the flow time of job j on system is the amount of time job j spends in the system. It is worth noting that flow time is equivalent to completion time when release time is zero.

Makespan ***(C_{Max}).*** The makespan is the completion time of the last job to leave the system.

Lateness ***(L_j).*** The lateness of job j is defined as $L_j = C_j - d_j$. The lateness is positive when completion time of job j is grater than its due date and it is negative when job j is completed early.

Tardiness ***(T_j).*** The tardiness of job j is defined as $T_j = Max(C_j - d_j, 0)$ $= Max(L_j, 0)$. The difference between lateness and tardiness is that tardiness is never negative, but lateness may be positive or negative.

2.3.1 Standard scheduling problem form:

The standard scheduling problem form, according to Graham et al [46], is $\alpha|\beta|\gamma$, where α describes the machine environment and contains a single entry, β defines the job characteristics or restrictive requirements and may contain no entries, a single entry, or multiple entries and γ indicates the objective function to be minimized and usually contains a single entry.

2.3.1.1 Machine environment (α)

There are many possible machine environments, specified in α:

Single machine *(1);* The case of a single machine is the simplest of all possible machine environments. Consideration of single machine problems is important because the results not only are usable in the single machine, but also provide a basis for heuristics for parallel machines or series machines. In fact, scheduling problems in more complicated environments often decompose into sub-problems that are single machine problems.

Identical machines in parallel *(Pm);* In this case there are m identical machines in parallel and job j requires a single operation which is possible to be processed on any one of the m machines. This is an important case from the practical point of view, since it is a common problem in the real world.

Flow shop *(fm);* There are m machines in series and each job requires one operation on each one of the m machines. All jobs have the same routing, i.e., each job should be processed from the first machine to the second machine and so on.

Open shop *(Om);* There are m machines and each job has to be processed on each of the m machines. The difference between the flow shop and the open shop is that in the open shop the processing time of some jobs may be zero and different jobs may have different routes.

Job shop *(Jm);* There are again m machines and each job has its own route. Each job visits each machine and in some cases it is allowed for a job to visit a machine more than once.

There is many other possible machine environments in the field of α which is cited in standard scheduling books.

2.3.1.2 Job characteristics (β)

There are also possible entries in respect of β. In the following some important possible entries in the β field that are used throughout this thesis are introduced:

Release date *(r_j);* When symbol r_j, i.e. release date, is presented in the β field, it means that the processing of job j cannot be started before its release date.

Preemptions *(prmp);* When symbol *prmp*, so called pre-emption, is presented in the β field, it means that it is allowed to interrupt the processing of a job at any time and put a different job on the machine. When a preempted job is put back on the machine, it only needs the machine for its remaining processing time.

Precedence constraints *(prec);* This situation may appear in single or parallel machine environments and imply that some jobs have *predecessors.* A job that has predecessors is not allowed to start its processing before the process of its predecessors is completed.

Breakdowns *(brkdwn).* When this symbol is presented in the β field, it implies that machine or machines are not available continuously.

Permutation *(prmu).* This symbol may appear in flow shop problems. When this symbol is presented in β field, it implies that the order in which the jobs meet the first machine will be maintained for other machines throughout the system.

2.3.1.3 Objective function (γ)

The objective to be minimized (γ) is always a function of completion time and depends on the schedule. The following are some examples of objective functions we will address through the thesis:

Minimizing Makespan *(C_{Max}).* As cited above the makespan is equivalent to the completion time of the last job to leave the system. Minimizing makespan is one of the most common objective functions in the literature and implies the high utilization of the machine.

Minimizing Maximum Lateness *(L_{Max}).* The maximum lateness shows the worst delay in respect to due dates.

Minimizing Total Completion time *($\sum C_j$).* As the term shows, the aim in this objective function is to minimise the sum of the completion times of all jobs in the

schedule. Minimizing the sum of the weighted completion times ($\sum W_j C_j$) is defined similarly.

Minimizing Total Flow time ($\sum F_j$). In the literature, the problem of minimizing sum of flow time is called the total flow time problem or F problem. Similarly the problem of scheduling to minimize weighted flow time is called F_w problem.

2.4 Scheduling with batching

Scheduling with batching is a subject that has recently received growing interest. The cause of this attention is that the processing of jobs in a batch or delivery of them in batch form may be cheaper or faster than processing or delivering them individually. The batching problem may occur in different domains. To state different types of batching problems we first introduce *family scheduling model.*

2.4.1 Family scheduling model

The number of jobs may be grouped to form a family based on their similarity. In fact in some cases these division of jobs to groups is necessary or worthwhile. One such case may occur where machines require setup times for processing jobs with different characteristics. The setup may be needed for washing the machine or for changing a tool. Consider the following example described in Monna and Potts [73]. An example is the process of production of different colours of paint on the same machine. A setup time is necessary for cleaning the machine whenever there is a colour change. It should be noted that here the setup time depends on both the colour being removed and the colour for which the machine is being prepared.

Classifying jobs into families gives the opportunity that just one setup time is required for a number of jobs belonging to the same family. However, a setup time is necessary when the schedule starts and every time the machine exchanges from processing jobs in one family to jobs in another family.

In family scheduling model, a *batch* is the maximum set of jobs that could be scheduled contiguously on a machine with one setup time. It is worth noting that large batches i.e. batches with too many jobs, give the advantage of high machine utilization because of the small number of setup time. On the other hand, processing

a large batch can delay the processing of important jobs that may belong to other families.

Batch delivery. Another situation where batching may produce efficiency is when the processed jobs have either to be delivered to another machine for further processing or to be delivered to customers. If there is a cost for dispatching jobs, then delivery of jobs in batch can reduce the total cost of transporting. In this situation a job may be processed but stays in the system to be delivered with other jobs which belong to one batch. It should be noted again that large batches have the advantage of reducing total delivery cost because the number of deliveries is small, but on the other hand, existing jobs in the system can increase the total flow time of system.

There are two types of family scheduling models depending on when the jobs become available: *batch availability* and *job availability* models.

Batch availability model is a model in which a job only becomes available when the complete batch to which it belongs has been processed. For example, this situation arises when the jobs in a batch are placed on a pallet, and the pallet is only moved from that machine when all the jobs are processed [79].

In contrast to batch availability, *job availability*, which is known in the literature as item availability, is a situation in which a job can be available immediately after it's processing is completed.

2.4.2 Batch processing model

Usually, in scheduling problems, it is assumed that a machine can process at most one job at a time. However, there are some machines that can process more than one job simultaneously. Hence, another situation for batching problem occurs when a group of jobs could be processed together. A group of jobs that could be processed together is called a *batch* and the machine that can process several jobs simultaneously is called *batching machine* or *batch processing machine.* Burn-in operations in semiconductor industries or heat treatment operations in metalworking are examples encountered with this situation [38]. The completion time of all jobs, which are processed together in a batch, is the same and it is equal to the greatest processing time of jobs within batch.

2.5 Algorithms and Complexity

Full treatment of the complex theory is outside the remit of this thesis. Here, we give a general definition of the theory and describe aspects relevant to the work in this thesis.

2.5.1 Definition

"An algorithm is a method for solving a class of problems. The complexity of an algorithm is the cost, measured in running time, or storage, or whatever units are relevant, of using the algorithm to solve one of these problems" [95].

More specifically, the time complexity of an algorithm is measured by running time while, the running time is the number of elementary operations required for implementation of the algorithm. The running time can be expressed as a function of the size of the input data, while the size of data usually measured in bits. Let n be a parameter that shows the size of problem and T be the function that shows the running time of an algorithm. By definition we say $T_w(n)$ is the worst-case running time of an algorithm if for any instance of problem with size n the running time is less than or equal to $T_w(n)$. The worst-case time complexity is usually stated by the $O, o, \Theta, \sim$, and Ω notation.

Let $f(n)$ and $g(n)$ be two functions of n. Each of the five notations cited above is intended to compare the rate of growth of f and g, or in other words they are some mathematical notations used to describe the asymptotic behaviour of functions.

The meaning of first two notations, which are used throughout this thesis, is described below:

O **Notation,** which is read "big O notation". We say $f(n) = O(g(n))$ or $f(n)$ is $O(g(n))$ if and only if there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$ where $n \to \infty$. When we say $f(n) = O(g(n))$, informally we are saying that g grows faster than f.

o **Notation** which is read "little o notation". We say $f(n) = o(g(n))$ or $f(n)$ is $o(g(n))$ if and only if $f(n)/g(n)$ exists and is equal to 0 for all $n \geq n_0$ where $n \to \infty$.

When we say $f(n) = o(g(n))$, informally we are saying that g grows much faster than f.

2.5.2 Easy and hard problems

The theory of time complexity leads us to classify problems in respect of complexity into two broad classes: *easy problems* and *hard problems*.

Easy problems are those that can be solved by polynomial-time algorithms. More clearly if the running time is at most a polynomial function of the amount of input data, then the calculation is an easy one" [95]. In contrast, the *hard problems* are those that are not likely to be solved in polynomial time and for which all known algorithms require exponential running time. In other words hard problems are problems for which there are no guaranteed solutions in the form of a polynomial function of the amount of input data.

It is worth noting that computation of some easy problems may take a very long time. But they are still in the group of easy problems from this point of view that there is at least a polynomial function of the amount of input data for these types of problems.

2.5.3 Decision problems

To classifying the complexity of problems in detail, we first need to introduce the decision problem. A decision problem is one where the required answer is either YES or NO. Therefore the decision problems are also called YES-NO problems. For every optimization problem one can create a decision problem [78]. Consider the following example. In problem $1||C_{\max}$ the makespan has to be minimized. Standard optimization form of this problem is given below:

Makespan-Optimization problem) a set of jobs is to be scheduled on a single machine while a processing time is associated to each job. The objective function is to find a sequence to minimize makespan, i.e. to minimize the completion time of the last job that leaves the machine.

We can transfer this problem to a decision problem in the following way:

Makespan-Decision problem) a set of jobs is to be scheduled on a single machine while a processing time is associated to each job. Does a schedule with a makespan less than given value K^* exist?

It is easy to see that if an optimization problem is solvable in a polynomial time then its decision problem is also solvable in a polynomial time. To do so, it is enough to create an algorithm for the makespan-optimization problem to determine an optimal value for makespan. We then can check to see whether the optimal value is less than K^* or not. The important point is that the converse is also true, i.e. it is proved that if decision problem of a given problem is solvable then its optimization problem is also solvable in a polynomial time [95].

2.5.4 Problem reduction

One of the most important rules in complexity theory is the concept of *problem reduction*. It is natural that very often one problem is a special case of another problem or equivalent to it or perhaps more general than it. Thus, an algorithm that works for one problem may also work with some minor change for another problem. "It is said that problem p reduces to problem p' if for any instance of p , an equivalent instance of p' can be constructed [78]".

For example it is obvious that problem $1||\sum C$ is a special case of problem $1||\sum w_j C_j$, meaning that if we can solve problem $1||\sum w_j C_j$ then problem $1||\sum C$ is also solvable with the same procedure. Using the terminology of complexity theory we say that problem $1||\sum C$ reduces to problem $1||\sum w_j C_j$ and it is denoted by $1||\sum C \propto 1||\sum w_j C_j$. It is worth noting that based on this concept a number of problems can be reduced to others. In addition, it implies that if we prove that a polynomial time algorithm exists for a problem, then the proof is extendable to all problems that can be reduced to that problem. However, it should be noted that the converse is not true. More specifically, if $p \propto p'$ and it is proved that a polynomial time algorithm exists for p' , then certainly a polynomial time algorithm exists for p , but if it is proved that a polynomial time algorithm exists for p , there is no guarantee that problem p' can be solved in polynomial time. On the other hand if it is proved that p is not solvable in a polynomial time, then p' is not certainly solvable in a polynomial time. However if p' is not solvable in a polynomial time it does not mean that we cannot find a polynomial time algorithm for p .

2.5.5 Complexity classes

As cited above the optimization problem and its decision problem are strongly related together. The complexity of problems according to related decision problem can be divided into four classes: classes $P, NP, NP-complete$ and $NP-hard$.

2.5.5.1 Class P

It is said that a problem belongs to class P if its related decision problem is in this class. Furthermore, A decision problem belongs to class p if there is an algorithm A that can solve every instance of the problem in a polynomial time.

2.5.5.2 Class NP

We define this class of problems according to Wilf [95]. It is said that a problem belongs to class NP if its decision problem belongs to class NP. Furthermore, a decision problem Q belongs to class NP if there is an algorithm A that satisfies the following:

1) For every YES instance I for which the answer is 'Yes', there is a polynomial length certification $C(I)$ such that when the pair $(I, C(I))$ are input to algorithm A it recognizes that I belongs to Q.

2) If I is some instance for which the answer is 'No' then there is no choice of certificate that will cause A to recognise I as a member of Q.

To understand this class more clearly it can be said "class NP is the class of decision problems for which it is easy to check the correctness of claimed answer with the aid of a little extra information [95]". Hence, it should be noted that in this definition we are not considering a method of solving the problem, but only indicating a method for checking the solution. More specifically, class P contains problems for which easy solutions could be found while class NP contains problems the solution of which can be easily checked, but for which the actual solutions may be very difficult to find.

2.5.5.3 Class $NP-complete$ and $NP-hard$

Like other classes, a problem belongs to class $NP-complete$ if its decision problem belongs to this class. Furthermore, a decision problem belongs to class $NP-complete$ if:

1) It belongs to class NP and,

2) Every problem in NP can be quickly reduced to it.

A problem that satisfies condition 2 but not necessarily condition 1 is said to be $NP-hard$. Thus, as a result the class $NP-hard$ can be introduced as a class of problems that are as hard as class $NP-complete$ or even harder still.

2.6 Literature review

In this section we consider briefly the relevant literature related to the work on the scheduling and batching problem. In the first section, the literature on single machine scheduling problem in absence of release date will be considered. In accordance with the first problem of this thesis the focus is on the batch availability problems while the objective function is minimizing the sum of weighted or in-weighed flow time (completion time). The second section, corresponding to the second problem of this thesis, considers the problem of minimizing the sum of flow time (completion time) in presence of release date. The literature on the problem of minimizing the sum of completion time on two-machine flowshop, appropriate to the third problem of this thesis, are surveyed in the third section. Finally, in the last section, the literature with the gain of scheduling with batch delivery are reviewed. It should be noted that there are only a few of papers with the same objective function of our work. Hence, in the last subsection we review papers on a combined problem, even their objective function is somehow different to that considered in this thesis.

2.6.1 Single machine without release date

One important class of batch availability models occurs when a machine requires setup time. There are two types of problems with setup time (cost). The first type is *sequence-independent,* under which setup depends only on the job to be processed. Under the second possible assumption, which is *sequence-dependent*, setup depends on both the job to be processed and the immediately preceding job. Such models

apply when jobs are partitioned into the families according to similarity, so that there is no need for setup or changeover when one job follows another of the same family; i.e., a setup is required only between different families and at the start of the schedule. Coffman et al. [27] consider the problem of minimizing total completion time with common setup time and prove that there exists an optimal schedule where the jobs are sequenced in Shortest Processing Time (SPT) order. Furthermore, they show that in the case where all families have a common setup time and the jobs are re-indexed, i.e. there is only one family and jobs are sequenced in SPT order, the problem is solvable by a backward dynamic programming algorithm that requires $O(n)$ time, where n is the number of jobs. However, most works on family scheduling under the batch availability assumption is concerned with minimizing the total weighted flow time. Albers and Brucker [3] have proved that this problem, even with common setup time for all families, is NP-hard but is solvable in $O(n \log n)$ time in the special case where all jobs have the same processing time and are sequenced in the non-increasing order of weights.

Masson and Anderson [70] propose a number of properties of the structure of the optimal solution of the problem of family scheduling under the batch availability. Furthermore, they derive a branch and bound algorithm for solving the problem of total weighted flow time. One of the most important aspects of their algorithm is to prove that in the optimal sequence batches are in order of non-decreasing WPT. Computational results imply that their algorithm is sufficient to handle problems up to 30 jobs.

Cheng, Chen and Oguz [18] consider a problem in which n jobs of T different types are to be processed on a single machine. The items are to be batched, such that the jobs in each batch are of the same type. A setup time is incurred between batches. The batches and the jobs within them are then to be sequenced to minimize the total weighted flow time. A dynamic programming algorithm that runs in $O(n^{T+1}/T^{T-1})$ is offered.

Webster and Baker [92] consider the problem of scheduling groups of jobs on a single machine. In respect of the problem of minimizing total flow times with batch availability, in a case when there is only one family, they analyse properties that are already established in the literature. They point out that the problem of minimizing

total flow times with multiple families is more complicated and the computational complexity of F problem remains an open question.

Liaee and Emmons [67] considered a variety of family scheduling problems with setup times under *group technology assumption* (GTA). Under GTA the job in a family must be scheduled contiguously. In the case when the objective function is minimizing total weighted completion times, they have shown that when the number of families is fixed, the optimal order of jobs can be found in $\sum_{k=1}^{K} O(n_k \log n_k) \leq O(n \log n)$. They have also investigated a case when the number of families is not fixed. In this case the problem is solvable only when the setup times are sequence independent, otherwise using the proof established by Rinnooy Kan [80] they have proved that even when each family contains only one job and all jobs have the same weight, problem is $NP-$hard.

Crauwels et al [31] propose a branch and bound algorithm for solving the problem. A lower bounding scheme based on a Lagrangian relaxation of the machine capacity constraint is derived that improves their algorithm in respect of the branch and bound algorithm which was provided by Masson and Anderson [70]. Also, a multiplier adjustment method to find values of the multipliers is used. They report that the computational experience with instances having up to 50 jobs shows that the lower bounds are effective in restricting the search.

A branch and bound algorithm based on a new lower bound is provided for the problem of minimizing weighted flow time of a set of jobs, which are divided into F families on a single machine by Dustall, Wirth and Baker [37]. The problem is considered under condition in which there is no need for setup time between jobs belonging to the same families, however a setup time is necessary whenever machine switch from processing a job in one family to a job in another family. Also an initial setup time is required when machine starts its processing, which is equal to the setup time of relevant family. They have also assumed that the setup times are sequence independent, that is the setup time between batches depends only upon the family being switched to. They have shown that their algorithm can solve instances with up to 70 jobs.

Potts and Kovalyov [79] offer a review of the scheduling with batching problem. Their special attention is focused on considering the efficiency of dynamic programming algorithms for solving this type of problems.

2.6.2 Single machine with release date

The classical problem of minimizing the sum of flow time in presence of release dates, i.e. $1|r_j|\sum F_j$ is unary NP-complete [66] and the preemptive version of the problem, i.e. $1|r_j, prmp|\sum F_j$ can be solved in polynomial time by the *Shortest Remaining Processing Time* (SRPT) rule [8]. Where all the jobs have identical release dates, the problem can be solved in $O(n\log n)$ time by applying the well-known *Shortest Processing Time* (SPT) [85]. The complexity of this problem has motivated the development of heuristic method [25], or some branch and bound based algorithms, for solving the problem [36]and [35]. Also for the same problem when the objective function is minimizing the total weighted completion time, Bianco and Ricciardelli [13], Dyer and Wolsey [39], Belouadah et al [12], Hariri and Potts [52]explored various branch and bound based algorithms.

Kellerer, Tautenhahn and Woeginger [61] have provided an approximation algorithm for solving the problem of scheduling n jobs with release date on a single machine to minimize the total flow time. Their algorithm is based on resolving of the preemption of the corresponding optimum preemptive schedule. They have presented the first approximation algorithm with a sublinear worst-case performance guarantee of $O(\sqrt{n})$. Then they have derived a lower bound for the problem and have proved that no polynomial time algorithm can have a worst-case performance guarantee of $O(n^{1/2-\varepsilon})$ with $\varepsilon > 0$.

Hall et al. [48] consider a variety of $NP-$hard scheduling problems in which the objective function is minimizing the weighted sum of completion times. They suggest two techniques to obtain a $\rho-approximation\ algorithm$ for this class of problems. The first technique is based on this observation that the lower bound given by the linear programming relaxation is always guaranteed to be a constant factor of the optimum value. The second technique is a generalization of designing on-line algorithms for minimizing total weighted completion time in presence of release dates. Their on-line algorithm relies "only on the existence of an (off-line) approximation algorithm for a problem that is closely related to finding a minimum-length schedule in that environment". In off-line algorithms the number of jobs and

their information are known in advance while in on-line algorithms the processing and release time of any job is known only after the job arrives [69]. The value of ρ facto, for several of scheduling problems is reported.

Philips, Stein and Wein [77] have explored an algorithm that converts preemptive schedule to non-preemptive schedule. They have reported that applying this algorithm to the problem of minimizing total completion time with release dates gives a 2-approximation algorithm for it. They have used the algorithm in a variety of problems including total weighted completion time in the presence of release dates on single and parallel machines.

The literature on improving ρ factor for the problem of minimizing the sum of completion times with release dates is extensive. In one of the latest articles on this topic, Goemans et al. [44] have reported the development of a randomized online algorithm whose worst-case bound is equal to 1.6853. Finally and more recently Chou [22] considers this problem under condition in which the processing time and weight of each job is a bounded positive number. In this case he proves that the asymptotic performance ratio of a simple online algorithm is one.

Kaminsky and Levi [60] consider the same problem and provide an algorithm that processes the jobs in order of shortest processing time among available jobs such that, at the completion time of any job, the next job to be scheduled is the shortest job among all those jobs that are released but not yet processed. They have shown that such an algorithm provides an asymptotic optimum value for the problem.

Ng, Cheng and Liu [76] have considered a serial batching scheduling problem in presence of release date and setup times to minimize the total completion time. They have investigated the situation in which the processing times of jobs are identical, there are precedence relations $\prec$ between jobs, and the jobs are to be processed in batches. A batch includes the number of jobs that have to be schedule on machine contiguously. The completion time of the last job in the batch is equal to completion time of the batch. A constant setup time will incur only when a new batch starts and it is assumed that a batch cannot be started before the maximum release date of the jobs within batch. They have provided a forward dynamic programming algorithm that solves the problem in an $O(n^5)$ time.

2.6.3 two-machine flow-shop

In respect of two-machine scheduling problem Garey, Johnson, and Sethi [43]have shown that the classical version of the problem, i.e. $F2||\sum F_j$ or $F2||\sum C_j$ is NP-complete. The flow time and completion time are equivalent when the jobs are ready at time zero. This problem was first studied by Ignall and Schrage [58]. They presented a branch and bound approach based on two lower bounds. The first lower bound is obtained by relaxing the constraint that does not allow the second machine to process more than one job at time, while the second lower bound is obtained by a similar relaxation on the first machine and also by relaxing the constraint that does not allow starting any processing on the first machine before time 0. Furthermore, for the second lower bound they use a redundant constraint which implies that the completion time of each job on the second machine is grater than or equal to processing time of that job on the second machine plus the shortest processing time of jobs on the first machine.

Conway, Maxwell and Miller [28] have shown that at least one optimal solution for this problem is *permutation* schedule without any idle time on the first machine. Van de Velde [88] developed a branch and bound method based on applying the Lagrangian relaxation on the constraint that requires for each job the second operation starts after the completion of the first operation, and showed that his lower bound dominated both bounds suggested by Ignall and Schrage.

Della Croce, Narayan and Tadei [34] consider several known lower bounds and present a new lower bound based on some new criteria. Computational result for instances up to 30 jobs indicate that when the new bound suggested by them applies jointly with the Van de Velde's lower bound, it gives the best performing lower bounding procedure.

Hoogeveen and Kawaguchi [55] analyse the worst-case behaviour of an algorithm presented by Gonzalez and Sahni [45] for the *m*-machine flowshop problem and in the case of two machines present a heuristic with the worst case bound of $2\beta/\alpha+\beta$, where α and β are defined as the minimum and maximum processing time of all operations respectively. Furthermore, they consider four special cases of the problem. In the case when all jobs on the first machine have equal processing time, they prove that problem is still $NP-$hard and then they present an approximation

algorithm with worst-case bound 4/3 that requires $O(n \log n)$ time, where n is the number of jobs. For three special cases, first: when the processing times of all jobs on the second machine are equal, second: when processing a job on the first machine takes no more time than its processing on the second machine, and third: while processing a job on the first machine takes no less time than its processing on the second machine, they prove that problem are solvable in polynomial time.

Della Croce, Ghirardi and Tadei [33] present a branch and bound method with offering two new dominance criteria for *pruning* some parts of solution space. They also present an enhancement of Van de Veld's lower bound by exploring sufficient conditions for the optimality of a given sequence when maximizing the Lagrangian dual problem. Computational tests on problems with up to 45 jobs are reported.

Can Akkan and Selcuk Karabati [2] present a new lower bound calculation scheme, which when integrated into a branch and bound algorithm that uses dominance criteria already established in the literature, can solve problems more efficiently. They have reported solving instances with as many as 60(45) jobs when processing times are uniformly distributed in the [1,10]([1,100]) range.

In respect of two-machine flowshop problem where setup times are separated from the processing time of jobs Aldowaisan and Allahverdi [4] consider a no-wait two-machine flowshop problem. The problem is characterized by a no-wait constraint where the jobs have to be processed continuously without waiting between or on consecutive machines. They provide some theorems and show that the problem with the objective of minimizing total flowtime for two special cases is optimally solvable. Both cases occur in the zero-buffer problem, i.e. the problem for which there is no intermediate buffers between machines. In the zero-buffer problems, when the processing of a job is finished on machine 1 but machine 2 is still busy, the job stay on machine 1 until machine 2 finishes the earlier job. The first special case occurs, when the setup of the next job on machine 1 cannot be started before the current job releases machine 1, while in the second case, the setup of the next job can be started immediately after the processing of the current job on machine 1 is completed. They also provide and test a heuristic algorithm for the generic problem.

Allahverdi [5] considers the problem of minimizing mean flow time in a two-machine flowshop with sequence-independent setup times. He shows that for two special cases the problem is optimally solvable. Let $s_{j,m}$ and $t_{j,m}$ identify the setup

time and processing time of job j for $j = 1,2,...,n$, respectively. For the first case, i.e. when $s_{j,1} + t_{j,1} \leq s_{j,2}$ for all jobs, he shows that sequencing the jobs in non-decreasing order of $s_{j,2} + t_{j,2}$ minimizes the mean flow time. For the second case, i.e. when the largest processing time for every job occurs on the second machine and $s_{j,1} + t_{j,1} \leq s_{h,1} + t_{h,1}$ for all j and h, he shows that sequencing the jobs in non-decreasing order of $s_{j,1} + t_{j,1}$ minimizes the mean flow time. He also develops a heuristic and branch and bounds for the general case.

Allahverdi, Gupta and Aldowaisan [6] present an extensive survey of scheduling problems involving setup.

In recent work on this topic, Wang and Cheng[91] consider a variety of special cases and show the optimal solutions of two cases. The first case occurs when the processing times of all jobs on both machines are equal to a constant t, and each job setup time on the second machine is less than that on the first machine. They show that for this case there exists an optimal schedule in which each batch consists of all jobs of a class, and the batches are sequenced in non-decreasing order of $s_{[i],1} / n_i$.

The second case appears when all jobs on both machines are equal to a constant t, and each job setup time on the second machine is no less than the sum of t and the setup time on the first machine. They show that for the second case there exists an optimal schedule in which each batch consists of all jobs of a class, and the batches are sequenced in non-decreasing order of $s_{[i],2} / n_i$. In the last 2 cases $s_{[i],1}$, $s_{[i],2}$ show setup time of the ith batch on machine 1 and 2 respectively and n_i shows the number of jobs in the ith batch. Based on the optimal properties of two last special cases and some other special cases they develop a heuristic and a branch and bound algorithm for the general case.

In another approach, by observing that in many practical situations scheduling problems may be involved with multiple objectives some authors have considered the problem of minimizing total flow time in a two-machine flowshop with minimum makespan. [23],[47].

2.6.4 Combined problem

Scheduling problems involving both machine scheduling and delivery cost appear to be rather complex, although they are more practical than those which involve just one of those factors. These types of combined optimization are often encountered when a real-world supply chain management is considered. Although there is a large body of research on the classical version of the problems, only a few articles address the combined optimization problem that seeks to coordinate machine scheduling with delivering jobs in batches. The complexity of some combined problems, such as makespan and completion times when the jobs are to be delivered after their processing time to customer or warehouse, is measured by Lee and Chen [64]. Hurink and Knust [57] consider a flow shop problem for minimizing makespan with transportation times, but they have assumed that a single robot, which can shift only one job at a time, does all the transportations.

Cheng and Covalyov [21] have considered a supply chain scheduling problem under the following situation. There is a supplier that has to produce some components for several manufacturers. The components are to be produced by the supplier serially on a production line. During non-productive breaks or shift changes, setup time can be ignored. However, each setup incurs a cost associated with the setup operations. The components are the same for each manufacturer and are delivered to them in batches of the same size. The batch delivery time depends on the manufacturer to whom the batch is delivered. The objective is to find a sequence of components such that the total setup cost is minimized, subject to maintaining continuous production for each manufacturer. The time at which the manufacturers start their production is given in advance. These times are equal to the production completion times of the previous production periods. It is proved that the problem is $NP-$hard. They have reduced the problem to a single machine scheduling problem with deadlines and job belonging to F families and developed an $O(N \log F)$ algorithm to find a feasible schedule for the problem, while N is the number of delivery batches. They have also provided a dynamic programming algorithm with $O(N^F / F^{F-2})$ running time to find the optimum schedule. In the case

where $F = 2$ and setup costs are unit, the time complexity of their algorithm reduces to $O(N)$ time.

Chang and Lee [16] have considered coordination of machine scheduling problem with job delivery. They investigated the situation in which jobs require different amount of storage space during delivery, a particular transportation time is associated with each delivery and all jobs delivered in one shipment to one customer have the same completion time. Furthermore there is only one vehicle available to deliver the finished jobs. They have shown that minimizing $C_{\max}$ for this problem, even in a simplified version i.e. when there is a single machine and the only one customer area, is intractable. In this paper $C_{\max}$ is defined as "the time when the vehicle finishes delivering the last batch to the customer site and return to the machine". Then they have provided heuristic method for the problem with the worst-case performance of $5/3$ and with a tight bound. Another heuristic is also provided for the case in which the finished jobs have to be delivered to two customers. It is proven that the error of their heuristic is no grater than 100% in any problem instance.

Hall and Potts [49] consider a variety of scheduling, batching and delivery problems. One of the problems identified by Hall and Potts is that of batching and sequencing on a single machine under the batch availability assumption in order to minimize the sum of flow times plus delivery costs. Using the idea of Albers and Brucker [3] for a similar problem, Hall and Potts provide a forward dynamic programming algorithm that has $O(n^{T+1})$ complexity, where T is the numbers of job families.

Another problem identified by Hall and Potts is that of batching and sequencing on a single machine under the batch availability assumption in order to minimize the sum of flow times plus delivery costs in presence of release date. Hall and Potts derive a forward dynamic programming algorithm for the problem under the assumptions of batch consistency. A supplier's batch schedule and a manufacturer's batch schedule are batch consistent if for each pair of jobs (i,h) and (j,h) that are processed by the supplier S and manufacturer M where $1 \le h \le H$, $1 \le i, j \le n_h$ and $i \ne j$, whenever job (i,h) is in a strictly earlier batch than job (j,h) in the supplier's batch schedule, then job (i,h) is in an earlier batch or in the same batch as job (j,h) in the manufacturer's batch schedule. They also make the further

assumption of SPT-batch consistency, in which jobs with the same release dates (that is, jobs delivered in one batch by the supplier) and for the same customer are processed by the manufacturer in SPT order. Hall and Potts have proved that the overall time complexity of their algorithm for finding an optimal schedule is $O(n^{3H})$ time while n, and H identify the number of jobs and customers respectively.

The third problem identified by Hall and Potts is that supplier and one manufacturer cooperate to solve a combined problem of minimizing the total system cost. Hall and Potts derive a forward dynamic programming algorithm for the problem under the assumptions of *total SPT within groups*. This implies that two stage-jobs for each customer are sequenced by both i.e. supplier and manufacturer, in SPT order according to the total processing time on supplier's machine and manufacturer's machine. They also make the further assumption than *total SPT within groups*, in which jobs for each of the manufacturers $M_2,...,M_g$ are sequenced in SPT order according to processing time on supplier's machine. They have proved that the overall time complexity of their algorithm for finding an optimal schedule is $O(n^{2G+7H-2})$ time while n, G and H identify the number of jobs, manufacturers and customers respectively.

Chapter 3

3. Minimizing the sum of flow times (completion times) from the view point of supplier

In this chapter we consider the problem of scheduling a set of jobs to be processed on a single machine by a supplier for delivery in batches to manufacturers for further processing. The problem is a natural extension of minimizing the sum of flow times by considering the possibility of delivering jobs in batches and introducing batch delivery costs. The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs. The extended problem arises in the context of coordination between machine scheduling and a distribution system in a supply chain network.

Structural properties of the problem are investigated and used to devise a branch and bound solution scheme. Computational experience shows significant improvements over an existing algorithm.

3.1 Introduction

Scheduling groups of jobs on a single machine is a subject that has recently received growing interest, due to the desire for exploiting economies of scale. The relevant models are called family scheduling models, for which two alternative assumptions may apply. The first is batch availability, under which all the jobs forming a batch become available for later processing or dispatch only when the

entire batch has been processed. Under the second possible assumption of job availability, a job becomes available once it has been processed. This work adopts the first assumption.

An extensive survey of family scheduling models is available in Webster and Baker [92]. Potts and Kovalyov [79] present a review of scheduling with batching. Their work is particularly focused on considering the efficiency of dynamic programming algorithms for solving this type of problems.

One important class of batch availability models occurs when a machine requires setup time. Such models apply when jobs are partitioned into families according to similarity, so that there is no need for setup or changeover when a job follows another of the same family; i.e., a setup is required only between different families and at the start of the schedule. In the case where there is just one family, the problem of minimizing total flow time (F) is solvable by a backward dynamic programming algorithm that requires $O(n \log n)$ time, where n is the number of jobs [79]. However, most work on family scheduling under the batch availability assumption is concerned with minimizing the total weighted flow time (F_w). Albers and Brucker [3] have proved that this problem is NP-hard, but is solvable in $O(n \log n)$ time in the special case where all jobs have the same processing time and are sequenced in the non-increasing order of weights.

Mason and Anderson [70] propose a number of properties of the structure of the optimal solution of the problem of family scheduling under the batch availability with minimizing total weighted flow time and derive a branch and bound algorithm for finding it. One of the most important aspects of their work is the proof that in the optimal sequence, batches are in order of non-decreasing Weighted Processing Time (WPT). Computational results show that their algorithm can deal efficiently with problems of up to 30 jobs.

Also a branch and bound method is provided by Crauwels et al. [31] that can solve problems of up to 50 jobs. The special feature of their algorithm, which is an improvement upon that of Masson and Anderson, is the use of a lower bound based on lagrangian relaxation of the machine capacity constraint.

Cheng, Chen and Oguz [18] consider a problem in which n jobs of T different types are to be processed on a single machine. The items are to be batched, such that

the jobs in each batch are of the same type. A setup time is incurred between batches. The batches and the jobs within them are then to be sequenced to minimize the total weighted flow time. A dynamic programming algorithm that runs in $O(n^{T+1}/T^{T-1})$ is offered. Allahverdi, Gupta and Aldowaisan offer an extensive survey of scheduling involving setup time [6].

Another important class of batch availability models occurs when the jobs are to be delivered to different customers or transferred to other machines in batches. In this case, no setup time is needed, but a delivery cost (delivery time) that depends on the customer is required for each batch. The problem is to batch and sequence batches and the jobs within them such that the sum of flow times plus delivery costs is minimized. As is cited in the last section this class of problems is important within the framework of supply chain management, and yet few works address it.

Cheng, Gordon and Kovalyov [19] consider a problem that arises when the objective is to minimize the sum of a function of the number of batches and job earliness penalties. Here the earliness of a job is defined as the difference between the batch delivery date and the job completion time. A relation between this problem and parallel machine scheduling is established, which in turn makes it possible to establishment of complexity results for the former problem based on known results for the latter problem.

Hall and Potts [49] consider the problem of batching and sequencing on a single machine under the batch availability assumption in order to minimize sum of flow times plus delivery costs. Using the idea of Albers and Brucker [3] for a similar problem, Hall and Potts provide a forward dynamic programming algorithm that has $O(n^{T+1})$ complexity, where T is the numbers of job families, each of which consists of all jobs destined for a particular customer.

In this chapter, we consider the problem of minimizing the sum of flow times plus delivery costs on a single machine under the batch availability assumption, study its structural properties, derive upper and lower bounds, offer a branch and bound scheme for solving it, and provide comparative results on efficiency.

3.2 Problem Definition

Let there be n jobs, which are to be delivered in batches to m manufacturers. These jobs are processed on a single machine that can process, at most, one job at a time and the processing time of a job i is p_i. Each job is produced for one manufacturer. A group of jobs forms a batch if they are all delivered to the appropriate manufacturer at the same time. Let D_j denote the non-negative cost of delivering a batch to manufacturer j. The objective is to minimize the total flow time or completion time plus delivery costs. This is a natural extension of the problem of minimizing total flow time to cater for coordination between scheduling and distribution in a supply chain network. The problem contains an additional term, which is the delivery cost for each manufacturer. Thus, using the standard classification scheme for scheduling problems [46], the objective function is $1||\sum F + \sum D_j k_j$, where k_j denotes the number of deliveries for manufacturer j.

3.3 Structural Properties

In this section, structural properties of the problem, used subsequently to derive upper and lower bounds, are analyzed.

Proposition 3.1. *For a set of batches, the sequence ordered by the Shortest Effective Batch Time (SEBT) is optimal in terms of total flow time, with batch effective time* $T_b = \frac{A_b}{\delta_b}$. *Here* A_b *is the total processing time of the batch and* δ_b *is the batch size (number of jobs in the batch), which could be equal to 1.*

Proof: (by contradiction). Consider a schedule S formed from a sequence that is not ordered by SEBT. In this schedule there must be at least two adjacent batches, say x followed by y, such that

$$\frac{A_x}{\delta_x} > \frac{A_y}{\delta_y}$$

Now consider the schedule S' formed by exchanging the positions of the two batches. Clearly, the flow times of all the batches preceding the pair under consideration will remain unaffected and so will the flow times of all the succeeding

batches. It is, therefore, sufficient to compare two total flow times: x followed by y with y followed by x.

For S: $A_x \delta_x + (A_x + A_y)\delta_y$

For S': $A_y \delta_y + (A_x + A_y)\delta_x$

Comparing terms, it is clear that the total flow time of S' is smaller than that of S. Thus, exchanging the positions of x and y reduces overall flow time. Proceeding in this manner, carrying out any beneficial pairwise exchanges will ultimately yield a schedule based on a sequence ordered by SEBT. □

Proposition 3.2. *Assume a partial schedule, where some batches have been formed on each machine, but no decision has yet been taken on batching the remaining 'un-batched' jobs. Here a lower bound on the sum of job flow times of an optimally completed schedule corresponds to the sum of job flow times in a schedule formed by considering each un-batched job as a single-job batch and sequencing all batches in the order of SEBT.*

Proof: When completing the schedule any batching of un-batched jobs will necessarily delay some jobs. Hence, considering each such job as a single-job batch ensures no delay. Thereafter, SEBT sequencing ensures that the resulting schedule minimizes total flow time by virtue of proposition 3.1. □

Since the jobs in a batch are all delivered at the batch delivery time, the order of the jobs within a batch is immaterial. However, further development of problem properties is simpler, if it is assumed that these jobs are ordered according to SPT. We will, therefore, make this assumption in what follows.

Proposition 3.3. *In an optimal solution, any batch destined for a manufacturer j that has $\delta_b > 1$ jobs will have the property that $(\delta_b - 1)p_l < D_j$, where l is the last job in the batch.*

Proof: (by contradiction). Consider a batch that does not have the indicated property. Removing the last job and delivering it in a batch of its own will decrease the overall

objective function by $(\delta_b - 1)p_l - D_j$, no matter where the batch happens to be in the sequence of batches. □

The following corollary then follows immediately.

Corollary 3.4. *In an optimal solution, any job that has a processing time greater than the batch delivery cost to the corresponding manufacturer, i.e., such that $p_i > D_j$, will form a single-job batch.*

Proposition 3.5. *If batches belonging to the same manufacturer are concatenated in the same order with which they occur in the optimal schedule, then the jobs will appear ordered by SPT.*

Proof: (by contradiction) Consider a schedule S that does not have the indicated property. Such a schedule will have two jobs i and k such that $p_k < p_i$ and k starts later than i. If both jobs belong to the same batch, then exchanging them will not affect the total flow time. However, if they belong to two separate batches, then exchanging them will reduce the flow time of the batch containing i and the flow times of all batches succeeding that of i and preceding that of k. This clearly reduces total flow time. Proceeding with carrying out any beneficial pairwise exchanges will ultimately yield a schedule that has the indicated property. It is also worth noting that the exchanges may yield opportunities for further batching, thereby reducing total delivery cost as well. □

The above proposition makes it possible to derive a lower bound on the number of batches destined for a manufacturer; a result that we present in the next proposition.

Proposition 3.6. *A lower bound on the number of batches destined for a manufacturer in an optimal solution can be found by the following greedy maximum batching algorithm: take the jobs destined for the manufacturer concerned in SPT order; if the job may be added to the current batch by virtue of proposition 3.3, then add it; else start a new batch.*

Proof: In accordance with proposition 3.5, jobs have to be taken in SPT order. Since each batch is augmented until it can take no more jobs, the number of batches is minimized. Moreover, since this is done while relaxing (forgetting) the constraints of interaction with jobs for other manufacturers due to batching, the number found is a lower bound, as claimed. □

In carrying out the search, we will be continually evaluating the worth of moves that add a job to a preceding batch for the same manufacturer. It is, therefore, important that this evaluation is carried out in a computationally efficient way. The following proposition helps to achieve that aim.

Proposition 3.7. *Let a job k be in position s_r to the right of a batch b, which is in position s_l in a SEBT sequence (in which job k constitutes a single-job batch). If k may be added to b by virtue of proposition 3.3, then the change in the sum of job flow times resulting from doing so, could be found by updating the contribution of the batches between s_l and s_r inclusive.*

Proof: Upon forming it, the new batch may have to be moved to restore the SEBT property. If it does not have to be moved, then the flow time of its old jobs, as well as the flow times of the batches in positions $s_l+1,...,s_r-1$, will increase by p_k, in addition to the decrease in the delivery time of k itself. If it has to move, then it will move to the right, since its effective batch time has increased, but to the left of position s_r+1, since its new effective batch time is $\leq p_k$. Thus, in this case also the contribution of batches in positions $l+1,...,s_l-1$, and batches in positions $s_r+1,...$ remains unchanged. □

In consequence of the above proposition, evaluating the worth of a job joining an earlier batch can be calculated efficiently, provided the delivery time, the number of jobs and the contribution of each batch to total flow time are kept updated.

Proposition 3.8. *In completing a partial schedule, where some batches have been formed, but no decision has been taken yet on batching the remaining 'un-batched'*

jobs, a 'batching penalty' Δ_k *, attaches to each un-batched job. This is because if the job is added to the last formed batch for the manufacturer concerned, total flow time will increase, and if a new batch is started with it, an additional batch delivery cost will be incurred. Moreover,* $\delta_l p_k \leq \Delta_k \leq D_j$ *, where* δ_l *is the number of jobs in the batch that it may join and* D_j *is the appropriate batch delivery cost.*

Proof: Consider a partial schedule. Add the jobs that have not been scheduled and order all in SEBT sequence. Let job in position k of the sequence be the first unscheduled job to the right of the last batch in position l for the same manufacturer. If the job in position k may not be added to batch in position l by virtue of proposition 3.4 or is not added to it, then a penalty equal to D_j will be incurred. Otherwise, total flow time will increase. To assess this increase, Recall from proposition 3.1 that :

$$\frac{A_l}{\delta_l} \leq \frac{A_{l+1}}{\delta_{l+1}} \leq \ldots \leq \frac{A_{k-1}}{\delta_{k-1}} \leq p_k ,$$

and that batches or jobs to the left of batch l and the right of job k will not be affected.

Four cases need to be distinguished:

Case 1 Job in position k follows batch in position l directly ($k = l + 1$):

$$\Delta_k = \delta_l p_k$$

Case 2 $k > l + 1$ and the newly formed batch remains in position l:

$$\Delta_k = \delta_l p_k + (\delta_{l+1} + \ldots \delta_{k-1})\, p_k - (A_{l+1} + \ldots + A_{k-1}) > \delta_l p_k$$

Case 3 $k > l + 1$ and the newly formed batch moves to the end of the subsequence, i.e., after the batch that was in position r - 1:

$$\Delta_k = \delta_l p_k + (A_{l+1} + \ldots + A_{k-1})\, \delta_l - (\delta_{l+1} + \ldots \delta_{k-1})\, A_l > \delta_l p_k$$

Case 4 $k > l + 1$ and the newly formed batch moves to a position in between, say, after the batch that used to be in position $l + \alpha$:

$$\Delta_k = \delta_l p_k + (A_{l+1} + \ldots + A_{l+\alpha})\, \delta_l + (\delta_{\alpha+1} + \ldots \delta_{k-1})\, p_k$$

$$- (\delta_{l+1} + \ldots \delta_{l+\alpha})\, A_l - (A_{\alpha+1} + \ldots + A_{k-1}) > \delta_l p_k .$$

Thus in each case $\Delta_k \geq \delta_l p_k$ which completes the proof. □

3.4 Branch and Bound Scheme

Search of the solution space is structured as a bivalent 0-1 search tree, where each node is partitioned into two, one indicating that a job is added to the last batch for the manufacturer concerned (1) and the other indicating the start of a new batch that could be a single-job batch (0). The tree is constructed in a depth-first fashion.

At the beginning, all jobs that have to form single-job batches by virtue of corollary 3.4 are identified and all their corresponding variables are set to 0 once.

Other components of the branch and bound scheme are presented in the following subsections.

3.4.1 Branching and ordering of variables

Variables are ordered in accordance with the SPT of the corresponding jobs. At each node of the decision tree, two tasks are performed. First, variables that have to be set to zero, because no batch for the manufacturer concerned has been formed or by virtue of proposition 3.3, are set to zero. Secondly, the first free variable (free variables are those that have not yet been committed to either zero or one) in the SPT sequence is set to one.

3.4.2 Fathoming and backtracking

A node is fathomed if:

1. It is a leaf node, i.e., all variables are fixed.
2. The lower bound exceeds or equals the incumbent upper bound.

Fathoming initiates backtracking to the first node associated with a variable whose value is 1; the value of this variable is then set to 0. If no such node is found, the search terminates.

3.4.3 Upper bounds

Providing a sharp, i.e., low, initial upper bound is critically important for enhancing the exclusion rate of the branch and bound algorithm, i.e., the rate with which nodes are fathomed. Hence, it is worth expending some computational effort to achieve that end. A number of upper bounds are, therefore, calculated and the sharpest is adopted.

***UB*1 --- Batch maximization heuristic:** Form single-batch jobs from all jobs that have to form such batches (proposition 3.3) and set them aside. Assemble all other jobs into batches corresponding to the minimum number of batches possible for each manufacturer, using the maximum batching algorithm (proposition 3.6). Arrange all batches in SEBT to minimize the corresponding total flow time (proposition 3.1). The total cost of the schedule thus formed then constitutes the incumbent upper bound, *UB**.

***UB*2 -- Multi-start greedy heuristic:**

For each manufacturer in turn **do**

Begin

Form all jobs into single-job batches and sequence the batches in SEBT order (which, in this case, is equivalent to SPT).

Treating the sequence as a circular array, start with the first batch belonging to the current manufacturer;

repeat

Scan forward until the first batch that may profitably be joined with the current batch is found. If found, join the two batches.

Move the newly formed batch forward to restore SEBT order, if necessary.

Move to the next job.

until a complete scan of all batches results in no improvement.

If the upper bound found is less than the incumbent, the former replaces the latter .

end.

3.4.4 Lower bounds

It is worth recalling that at each node of the decision tree, if, in view of the batching decisions already taken, a job has to start a new batch, then the partial solution is immediately augmented by a batch starting with that job.

At each node of the decision tree, a lower bound on total flow time is calculated in accordance with proposition 3.2. Additionally, batch delivery costs are added for each batch already formed.

Furthermore, a lower bound on the batching penalties is calculated by applying the logic of proposition 3.8 in the following way. The first un-batched job of a manufacturer, k, will either start a new batch or join the last batch. It would,

therefore, attract a lower bound on its batching penalty = $\delta_l p_k$, where δ_l is the number of jobs in the last batch of the manufacturer concerned. Since for each subsequent job, we do not know the number of jobs in the batch that it may join in the optimal completion of the current partial solution, each such job would attract a lower bound on its batching penalty = p_k (i.e., the lowest batching penalty that it may incur would be if it joined a single-job batch).

At the initial stages of the search, opportunities arise for tightening the overall lower bound still further by considering proposition 3.6. Assume that the number of batches already formed for a particular manufacturer in the current partial solution is b_k and that the minimum number of batches is $b_{\min}$. It would then be possible to raise the sum of lower bounds on batching penalties by identifying the $b_{\min} - b_k$ highest individual lower bounds on batching penalties and replacing each by the batch delivery cost.

The overall lower bound is then the sum of the lower bound on the total flow time, batch delivery costs of the batches already formed and the sum of the lower bounds on the batching penalties for the un-batched jobs.

3.4.5 Numerical example

Consider the following two- manufacturer problem, with delivery costs of 11 and 8 respectively:

	Job Processing Times			
	Job 1	Job 2	Job 3	Job 4
Manufacturer 1	3	4	5	10
Manufacturer 2	2	6	7	—

Let (ij) denote the ith job destined for the jth manufacturer.

Initial lower bound:

The lower bound on total flow time, *LBF*, corresponding to flow time under SPT = 114. Batch delivery costs, *LBD*, is 19 (we have to start a batch for each manufacturer).

The lower bound on batching penalties, *LBB* is: (4+5+11) + (6+8) = 34, where the numbers within the first (second) bracket correspond to the penalties incurred by

each job of the first (second) manufacturer in turn. Note that the 11 for the first manufacturer and the 8 for the second manufacturer result from substituting the initial lower bounds on the batching penalty, which are equal to 10 and 7 respectively, by the batch delivery costs, by virtue of the fact that the minimum number of batches for either manufacturer is two.

$LB = LBF + LBD + LBB = 114 + 19 + 34 = 167$

Initial upper bound:

Applying SEBT to the 4 batches yielded by maximum batching (proposition 3.6) leads to the following schedule: | (12) (22) |, | (11) (21) (31) |, | (32)|, | (41) |, with a total flow time of 140. Adding total batch delivery costs gives $UB^* = 178$.

Applying the multi-start heuristic starting with manufacturer 2 yields an inferior upper bound.

Applying the multi-start heuristic starting with manufacturer 1 yields the following schedules, successively:

| (12) | (11) | (21) | (22) | (32) | (31) |,
| (12) | (11)(21) | (31) | (22) | (32) | (41) |,
| (12) | (11)(21) (31) | (22)| (32) | (41) |,

Attempting to add (41) to the last batch for manufacturer 1 proves unprofitable,
Attempting to batch (12) and (22) proves unprofitable,

| (12) | (11)(21) (31) | (22) (32) | (41) |,

The last schedule has an objective function value of 173. Therefore, $UB^* = 173$.

Branch and Bound

Each item below represents a decision node. Search of the decision tree is carried out depth first. S denotes the partial solution (batching decisions) at each node.

0. S_0 = |(11) |, | (12)|; LBF = 114, LBD = 19, LBB = 4+5+11+6+8=34, and LB = 167.

1. S_1 = |(12)|, | (11) (21) |; LBF = 118, LBD = 19, LBB = 2*5+11+6+8=35, and LB = 172.

2. S_2 = |(12)|, | (11) (21) (31) |, |(41)|; LBF = 128, LBD = 30, LBB = 6+8=14, and LB = 172.

3. S_3 = |(12) (22)|, | (11) (21) (31) |, | (32)|, |(41)|; LBF = 140, LBD = 38, LBB =0, and LB = 178. Since LB > UB*, backtrack.

4. S_4 = |(12)|, | (11) (21) (31) |, |(22)|, |(41)|; LBF = 128, LBD = 38, LBB = 7, and LB = 173. Since LB = UB*, backtrack.

5. S_5 = |(12)|, | (11) (21) |, | (31)|; LBF = 118, LBD = 30, LBB = 10+6+8=24, and LB = 172.

6. S_6 = | (11)(21) |, |(12) (22)|, |(13)|, |(32)|; LBF = 128, LBD = 38, LBB = 10, and LB = 176. Since LB > UB*, backtrack.

7. S_7 = |(12)|, | (11)(21)|, |(31)|, | (22)|; LBF = 118, LBD = 38, LBB = 10+7=17, and LB = 173. Since LB = UB*, backtrack.

8. S_8 = |(12)|, | (11)|, |(21) |; LBF = 114, LBD = 30, LBB = 5+10+6+8=29, and LB = 173. Since LB = UB*, backtrack.

9. Search Completed.

Thus the initial upper bound is optimal.

3.5 Computational Results

In the absence of benchmark test problem instances we resorted to generating two sets of problem instances to test the branch and bound algorithm (B&B). Furthermore, we compared the performance of B&B algorithm with the dynamic programming algorithm (DP) of Hall and Potts, which is the only other available algorithm for the problem under consideration.

For each set, three subsets were generated; one with 4 manufacturers, another with 8 and the third with 12. In each subset, the number of jobs was varied up to a total of 50. For the first set, in each instance, jobs were divided equally among manufacturers, with any remainder assigned to as many manufacturers as needed. For the second set, jobs were randomly distributed among manufacturers, with each

being assigned at least two jobs. For both sets, processing times were randomly generated integers from the uniform distribution defined on [1,100]. To ensure that results were representative, 5 instances were generated for each combination of number of jobs-number of manufacturers. Each of the running times in tables below represents the average over the appropriate five instances.

Since interaction between batch delivery costs and job processing times may affect problem hardness, we generated two classes of problems. In class *A*, all job processing times are less than the delivery cost of the relevant manufacturer, while in class *B* it is possible randomly that some jobs have a processing time greater than the delivery cost.

The computational experiments were run on a Pentium 4 computer with 2.40 GHz of CPU and 512 MB of RAM. Both B&B and DP were coded in C++.

Comparative results are shown in tables 1 to 12. Tables 1 to 6 show the results for problem instances when the jobs are uniformly distributed among manufacturers and tables 7 to 12 show the results for problem instances when the jobs are randomly distributed among manufacturers.

As can be clearly seen, the DP algorithm has advantage over B&B only for some class *A* problem instances having 4 manufacturers. For all other instances, B&B is more efficient by far. This is due to the time complexity of the DP algorithm, which can be clearly observed in the escalation of computing time with the increase in the number of jobs. On the other hand, the efficiency of the B&B, which enables it to solve most of problem instances in less than one second, is attributable to the effectiveness of the initial upper bound and the sharpness of the lower bounds. Careful analysis of the structural properties of the problem proved crucial in forging both sub-algorithms (upper bound and lower bound).

Furthermore, the complexity of the dynamic programming algorithm is such that for the same number of jobs, problem instances with a larger number of manufacturers are more difficult. Interestingly, the opposite is true for the branch-and-bound algorithm, as revealed by the solution times. This is so because when the number of manufacturers is large, the algorithm is in effect searching over a larger number of smaller subsequences, with much smaller numbers of possible permutations. However, it is expected that by increasing the number of jobs for the same number of manufacturers the running times increase. In this regard, we define a

critical point (CP), which shows the number of jobs for which the B&B algorithm can still solve problem instances efficiently. It is worth noting that CP is not the maximum number of jobs that can be solved by B&B, but only the number of jobs after which the running time increases critically. The CP for each combination of manufacturer-jobs is shown in the caption of diagrams 1 to 12.

Table 1: Running times for problem instances with jobs uniformly distributed among 4 Manufacturers, Class *A*.

Number of Manufacturers	Number of jobs	Running time (ms)	
		DP	B&B
4	13	10	12
4	18	10	16
4	23	20	18.2
4	28	30	44
4	33	70	77.5
4	38	120	268.4
4	43	200	1390
4	48	330	11470.6
4	49	360	12295.25
4	50	391	17837.6

Diagram 1: Running times versus number of jobs for problem instances with jobs uniformly distributed among 4 Manufacturers, Class *A*. CP = 45.

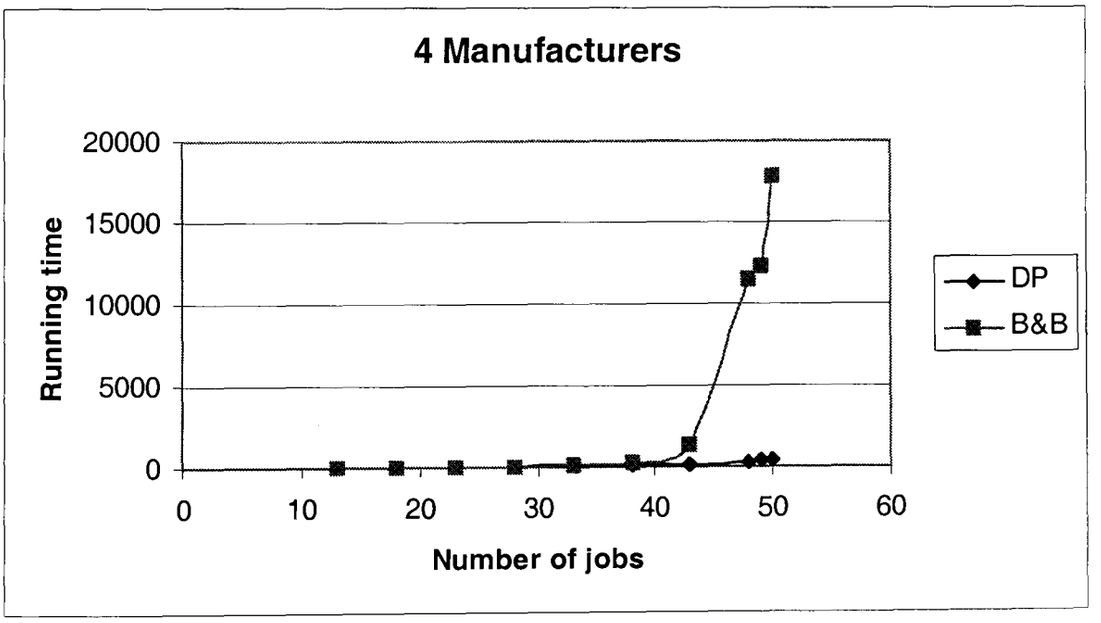


Table 2: Running times for problem instances with jobs uniformly distributed among 8 Manufacturers, Class *A*.

Number of Manufacturers	Number of jobs	Running time (ms)	
		DP	B&B
8	16	60	14
8	21	260	14
8	26	1122	32
8	31	4016	34.2
8	36	11908	88
8	41	32547	302.6
8	46	79064	1019.4
8	47	93454	1618.4
8	48	111110	5181.4
8	49	129466	1370
8	50	150837	2551.6

Diagram 2: Running times versus number of jobs for problem instances with jobs uniformly distributed among 8 Manufacturers, Class *A*. CP = 63.

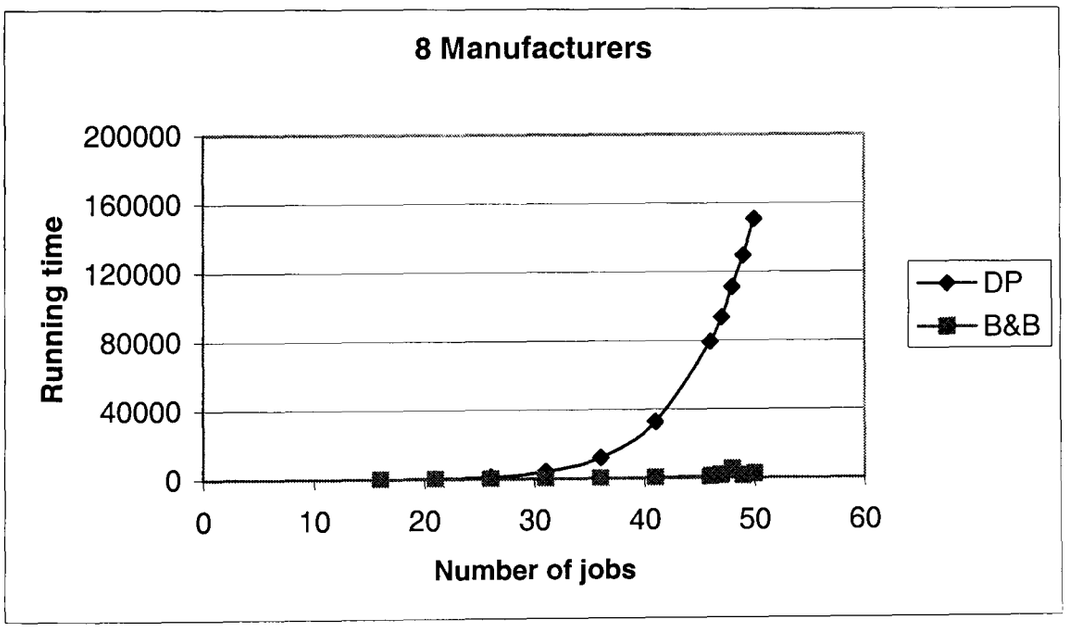


Table 3: Running times for problem instances with jobs uniformly distributed among 12 Manufacturers, Class *A*. *: Computer gives up for lack of sufficient.

Number of Manufacturers	Number of jobs	Running time (ms)	
		DP	B&B
12	24	7681	14
12	26	14681	26
12	28	27900	24
12	30	52825	34
12	32	100074	24
12	34	188250	54
12	36	353428	52
12	38	578622	74.2
12	40	952931	106.2
12	42	1618528	154.2
12	44	2742203	170.2
12	46	4908498	344.6
12	47	6504713	576.8
12	48	8697186	536.8
12	49	*	739.6
12	50	*	647

Table 3: Running times versus number of jobs for problem instances with jobs uniformly distributed among 12 Manufacturers, Class *A*. CP = 78.

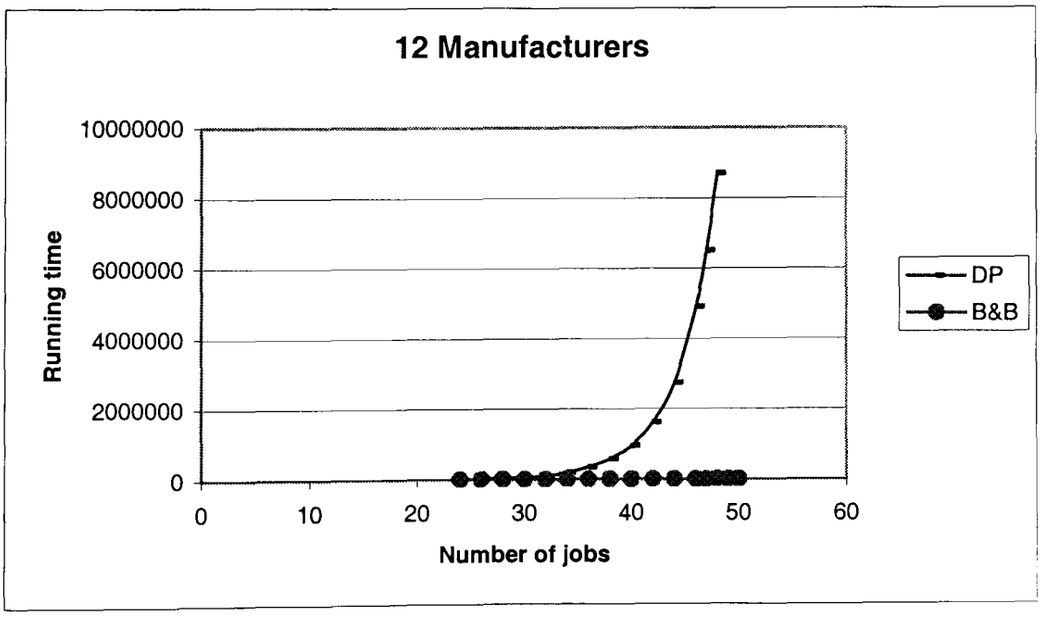


Table 4: Running times for problem instances with jobs uniformly distributed among 4 Manufacturers, Class *B*.

Number of Manufacturers	Number of jobs	Running time (ms)	
		DP	B&B
4	13	10	6
4	18	20	8
4	23	20	12
4	28	40	24
4	33	70	24
4	38	120	52
4	43	200	68.2
4	48	340	230.25
4	49	361	95.25
4	50	401	118.2

Diagram 4: Running times versus number of jobs for problem instances with jobs uniformly distributed among 4 Manufacturers, Class *B*. CP = 90.

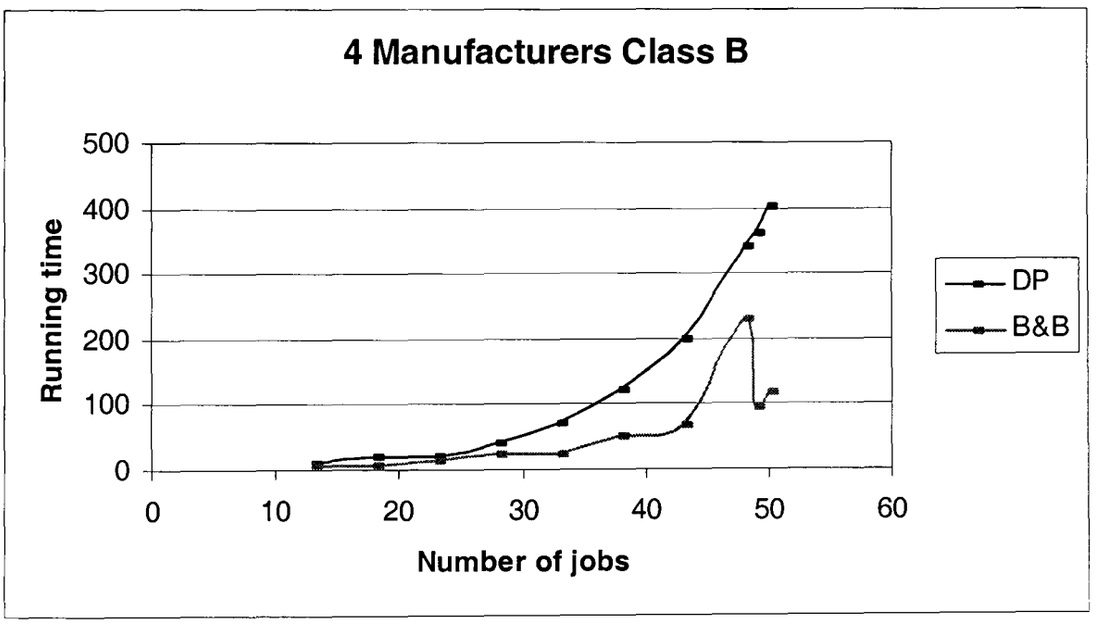


Table 5: Running times for problem instances with jobs uniformly distributed among 8 Manufacturers, Class *B*.

Number of Manufacturers	Number of jobs	Running time (ms)	
		DP	B&B
8	16	60	4
8	21	251	8
8	26	1151	6
8	31	3756	32
8	36	10845	22
8	41	31195	34.2
8	46	74637	50
8	47	91381	54
8	48	101086	64.2
8	49	127874	76.2
8	50	137888	124.2

Diagram 5: Running times versus number of jobs for problem instances with jobs uniformly distributed among 8 Manufacturers, Class *B*. CP = 110.

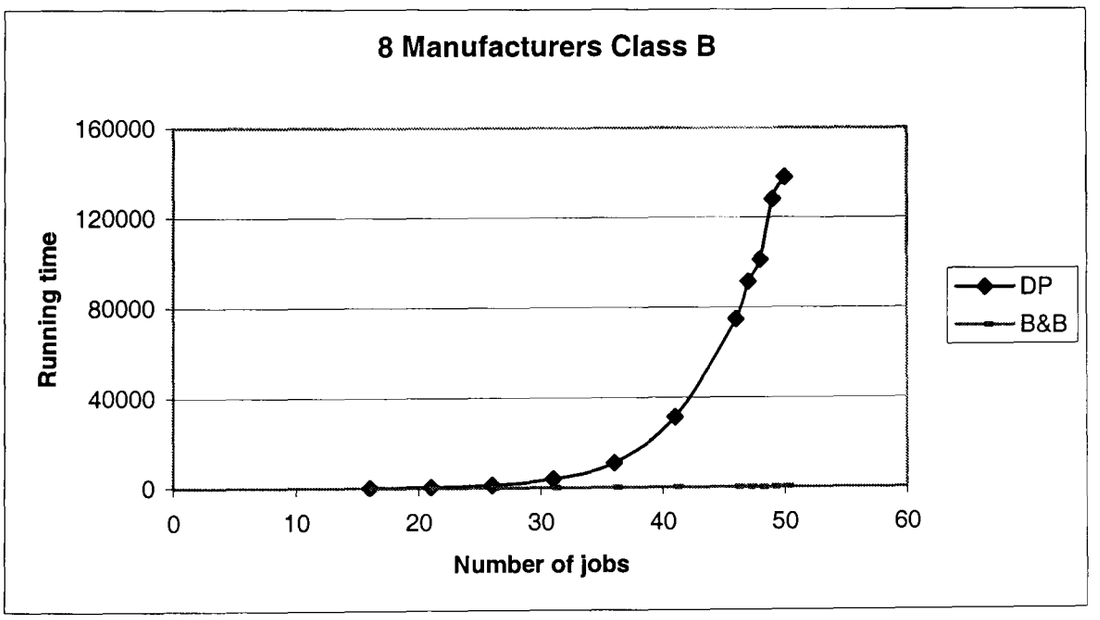


Table 6: Running times for problem instances with jobs uniformly distributed among 12 Manufacturers, Class *B*. *: Computer gives up for lack of sufficient.

Number of Manufacturers	Number of jobs	Running time (ms)	
		DP	B&B
12	24	6680	8
12	26	13199	12
12	28	24486	8
12	30	52976	12
12	32	96809	18
12	34	167281	16
12	36	309455	12
12	38	536131	12
12	40	859496	28
12	42	1479187	26
12	44	2514616	24
12	46	4674362	38.2
12	47	6206715	50
12	48	7383347	34
12	49	*	74.2
12	50	*	134.2

Diagram 6: Running times versus number of jobs for problem instances with jobs uniformly distributed among 12 Manufacturers, Class *B*. CP = 140.

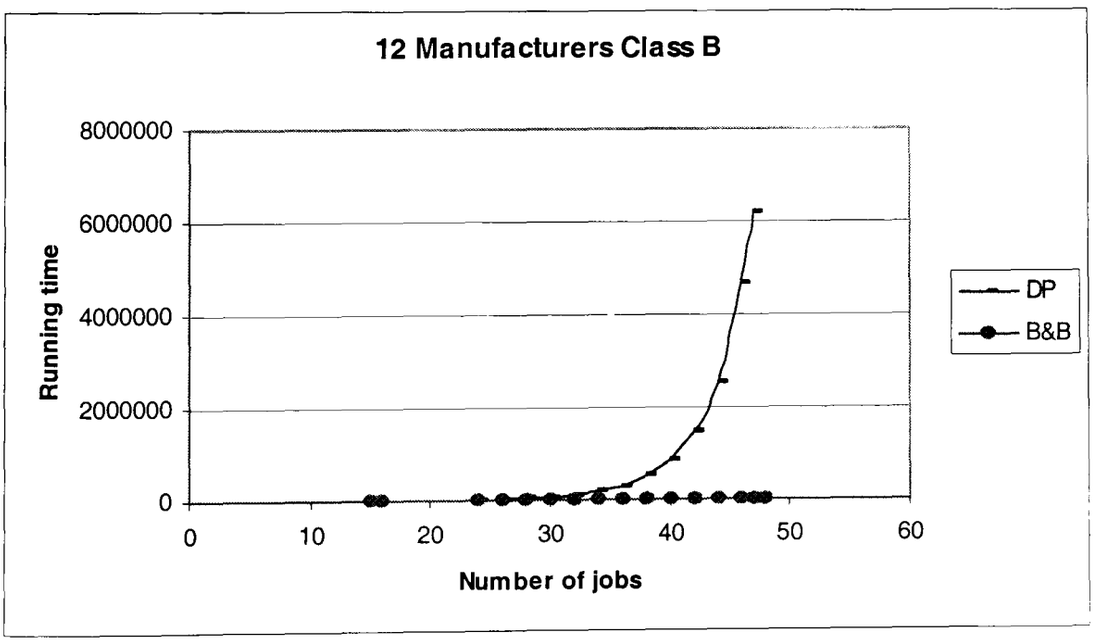


Table 7: Running times for problem instances with jobs randomly distributed among 4 Manufacturers, Class *A*.

Number of Manufacturers	Number of jobs	Running Time (ms)	
		DP	B&B
4	13	16.2	12
4	18	18	18.2
4	23	28	24
4	28	44	74
4	33	76.2	118.2
4	38	130.2	562.8
4	43	210.4	2317.4
4	48	346.4	11030
4	49	366.6	17252.8
4	50	388.6	10749.4

Diagram 7: Running times versus number of jobs for problem instances with jobs randomly distributed among 4 Manufacturers, Class *A*. CP = 48.

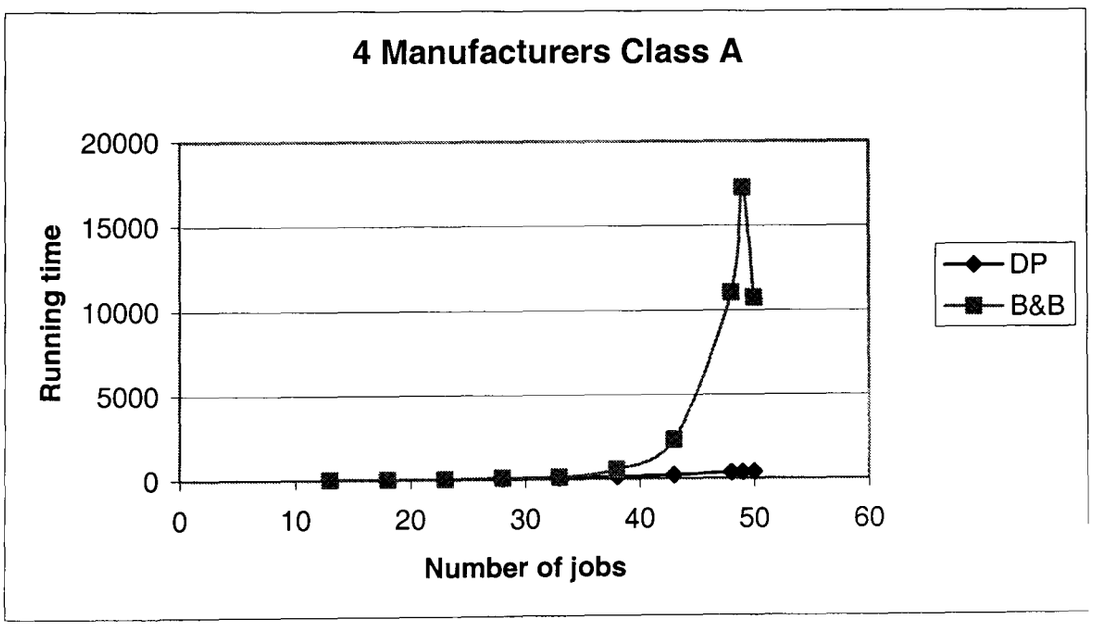


Table 8: Running times for problem instances with jobs randomly distributed among 8 Manufacturers, Class *A*.

Number of Manufacturers	Number of jobs	Running Time (ms)	
		DP	B&B
8	16	74.2	20
8	21	284.4	20
8	26	1089.6	26
8	31	3947.8	32.2
8	36	10825.4	82
8	41	27485.4	382.6
8	46	61096	2067
8	47	79007.6	987.4
8	48	89288.4	2411.4
8	49	110617	1896.8
8	50	126720.2	5373.8

Diagram 8: Running times versus number of jobs for problem instances with jobs randomly distributed among 8 Manufacturers, Class *A*. CP = 65.

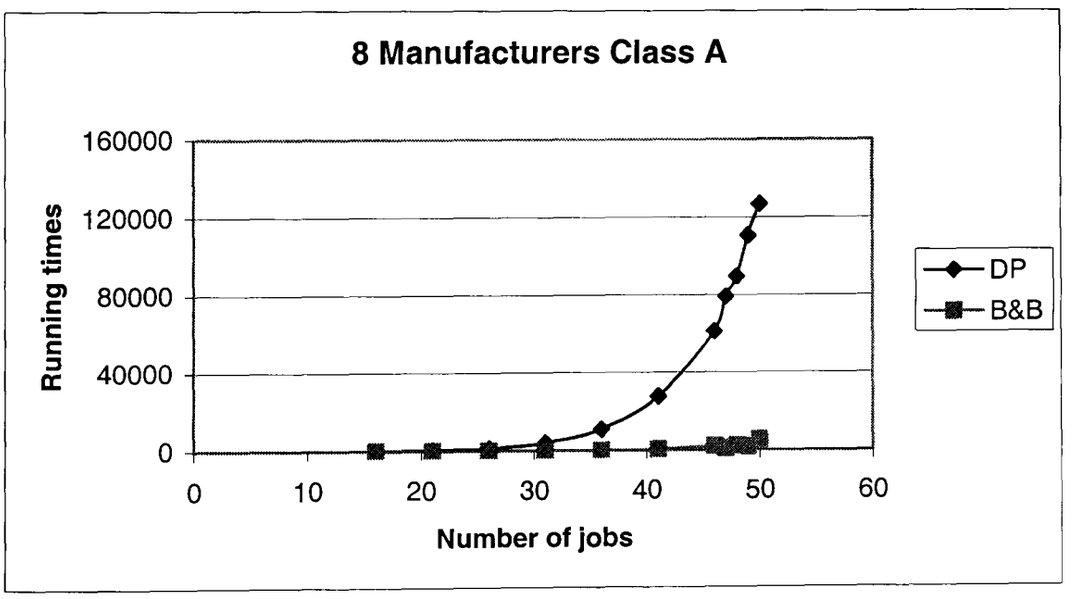


Table 9: Running times for problem instances with jobs randomly distributed among 12 Manufacturers, Class *A*. *: Computer gives up for lack of sufficient.

Number of ufacturers	Number of jobs	Running Time (ms)	
		DP	B&B
12	24	8512.4	28.2
12	26	16163.2	30
12	28	30179.4	24
12	30	54314	34
12	32	95926	32.2
12	34	163098	54
12	36	291997.8	54
12	38	476263	50
12	40	646781.8	100.2
12	42	1414776	206.4
12	44	1882855	436.4
12	46	2903481	528.8
12	47	4269381	380.6
12	48	6298261	660.8
12	49	*	1019.6
12	50	*	1650.4

Diagram 9: Running times versus number of jobs for problem instances with jobs randomly distributed among 12 Manufacturers, Class *A*. CP = 83.

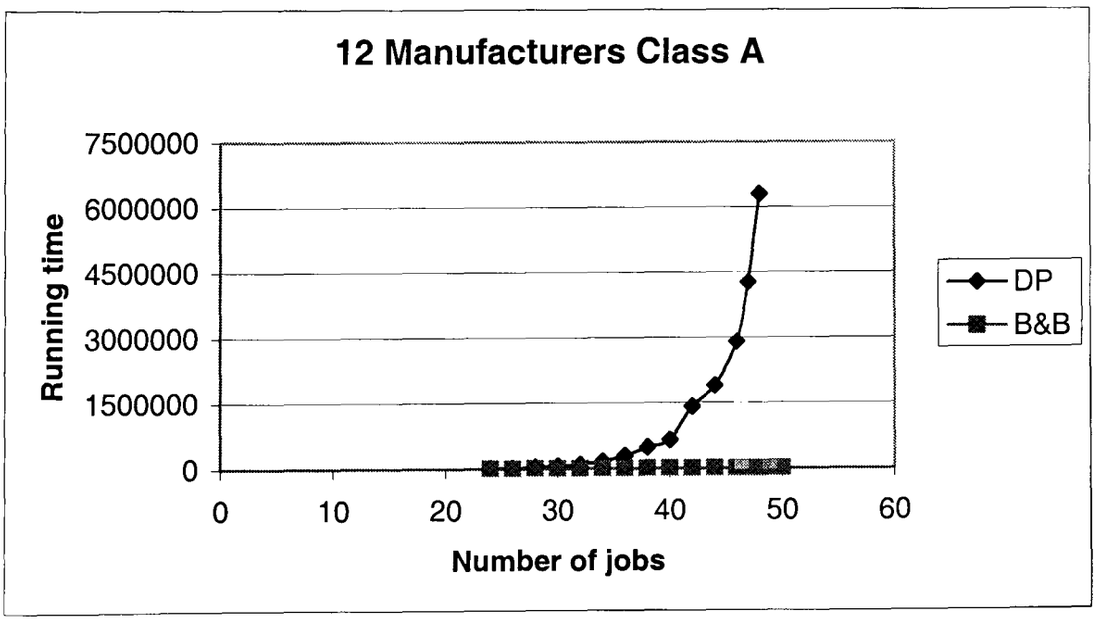


Table 10: Running times for problem instances with jobs randomly distributed among 4 Manufacturers, Class *B*.

Number of Manufacturers	Number of jobs	Running Time (ms)	
		DP	B&B
4	13	16	12
4	18	24.2	16
4	23	28	14
4	28	46	24.2
4	33	80.2	28
4	38	130	56
4	43	210	186.4
4	48	302.6	68
4	49	332.4	124
4	50	392.6	360.75

Diagram 10: Running times versus number of jobs for problem instances with jobs randomly distributed among 4 Manufacturers, Class *B*. CP = 90.

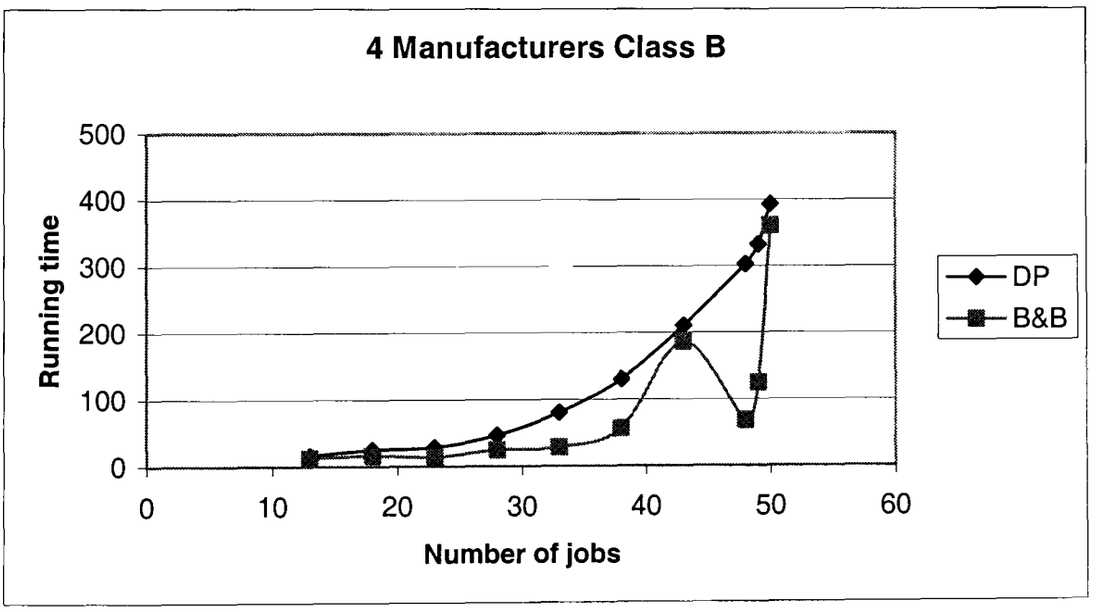


Table 11: Running times for problem instances with jobs randomly distributed among 8 Manufacturers, Class *B*.

Number of Manufacturers	Number of jobs	Running Time (ms)	
		DP	B&B
8	16	72.2	6
8	21	248.4	10
8	26	879.2	10
8	31	2693.8	34
8	36	9285.4	18
8	41	24162.8	26
8	46	54394.2	140.2
8	47	67236.8	150.2
8	48	82596.6	414.6
8	49	92390.8	130.2
8	50	114985.4	150.2

Diagram 11: Running times versus number of jobs for problem instances with jobs randomly distributed among 8 Manufacturers, Class *B*. CP = 112.

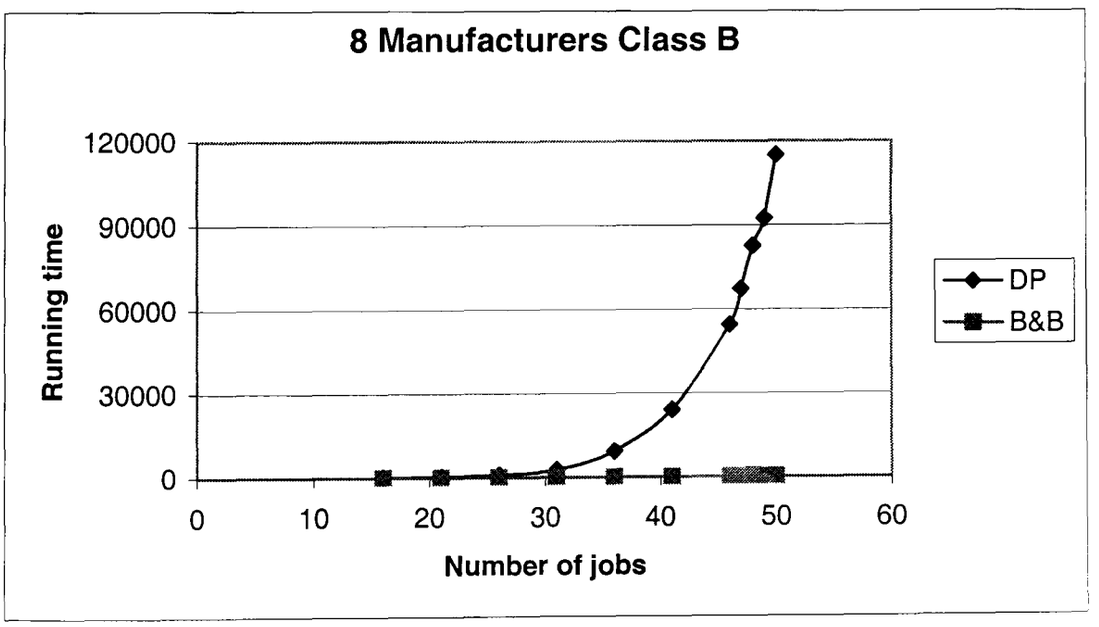
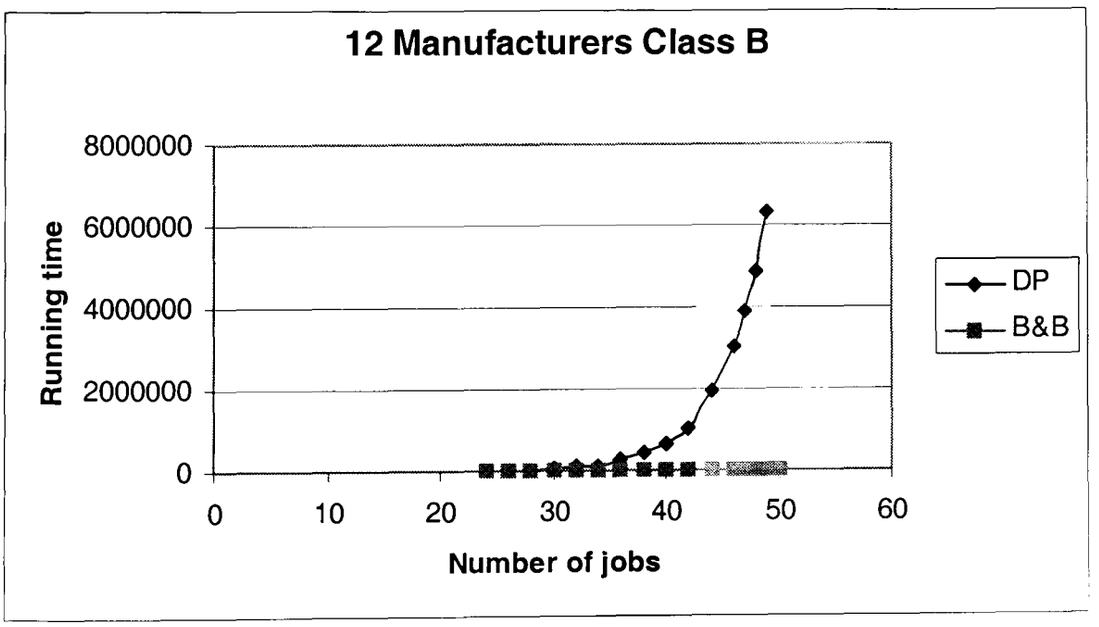


Table 12: Running times for problem instances with jobs randomly distributed among 12 Manufacturers, Class *B*. *: Computer gives up for lack of sufficient.

Number of Manufacturers	Number of jobs	Running Time (ms)	
		DP	B&B
12	24	6916	8
12	26	13239	8
12	28	24223	12
12	30	43027.8	6
12	32	81238.8	20
12	34	133303.8	12.2
12	36	248445.2	22
12	38	427685	22
12	40	662110.2	18
12	42	1006627	44
12	44	1958536	26
12	46	3027516	70.2
12	47	3871723	32
12	48	4864320	34
12	49	6327414	62
12	50	*	94.2

Diagram 12: Running times versus number of jobs for problem instances with jobs randomly distributed among 12 Manufacturers, Class *B*. CP = 150.



Effectiveness of the upper bound makes it possible to use it as a fast heuristic. Table 13 shows that on average it produces solutions that are within 0.23% of the optimum (relative error = $(\frac{UB}{Optimal}-1)$). Moreover, the error is smaller for larger instances. However, it may be argued that the efficiency of the B&B obviates the need for a heuristic solution.

Table 13: Average percentage Relative Error.

	Uniform Distribution of Jobs		Random Distribution of Jobs	
Manufacturers	Class A	Class B	Class A	Class B
4	0.15	0.07	0.23	0.08
8	0.09	0.02	0.09	0.03
12	0.04	0.01	0.05	0.01

3.6 Conclusion

A branch and bound algorithm for scheduling a set of jobs to be processed on a single machine for delivery in batches to manufacturers, or to other machines, for further processing has been presented. This problem is a natural extension of minimizing the sum of flow times by considering the possibility of delivering jobs in batches and introducing batch delivery costs. The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs. The extended problem arises in the context of coordination between machine scheduling and a distribution system in a supply chain network.

The branch and bound algorithm proved to be very efficient. Indeed, it proved to be far more efficient than the only existing algorithm for solving the problem, which is based on dynamic programming. This efficiency is attributable to the sharpness of the lower bounds derived, in addition to the high quality of an initial upper bound found using an effective heuristic.

Both lower bound and the upper bound were derived from a careful analysis of the structural properties of the problem.

Chapter 4

4. Minimizing the sum of flow times (completion times) from the view point of manufacturer

In the last chapter we considered the situation that supplier processes the jobs and delivers them in batches to manufacturers. The time at which each job is delivered to manufacturers defines a release time within which manufacturers cannot start processing of the jobs. Hence, in this chapter we consider the problem of scheduling a set of jobs to be processed on a single machine by manufacturer for delivery in batches to customers, while the jobs are available for processing at their release times. The problem is a natural extension of minimizing the sum of flow times with job release times by considering the possibility of delivering jobs in batches and introducing batch delivery costs. The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs. The extended problem arises in the context of coordination between machine scheduling and a distribution system in a supply chain network.

Structural properties of the problem are investigated and used to devise a branch and bound solution scheme. Computational experience shows significant improvements over an existing algorithm.

4.1 Introduction

Single-machine scheduling has been studied extensively with different objective functions. In this chapter, we describe a model for minimizing total flow times plus

delivery cost for a set of jobs that are to be processed on a single machine for delivery in batches to customers when each job is available at its release time. This situation may arise in a supply chain networks when jobs arrive at different times to a manufacturer who processes them for delivery to some customers or transfer them in batches to other machines for further processing. In this model, the time at which each job, j, is delivered from supplier to manufacturer defines a release time from the viewpoint of manufacturer. This is the recognition version of the $1|r_j|\sum F_j$ classical problem with an additional term. The classical problem is unary NP-complete [66] and its preemptive version, $1|r_j, prmp|\sum F_j$, can be solved in polynomial time by the Shortest Remaining Processing Time (SRPT) rule [8]. Also, where all the jobs have identical release times, the problem can be solved in $O(n\log n)$ time by applying the well-known Shortest Processing Time (SPT) [85]. The complexity of this problem has motivated the development of a heuristic method [25] or some branch and bound based algorithms for solving the problem [36], and [35]. Also for the same problem when the objective function is minimizing the total weighted completion time, various branch and bound based algorithms are explored [13], [39], [12] and [52].

Problems that address an optimal value that includes both machine scheduling and delivery costs appear to be rather complex, though they are more practical than those that involve just one of these two factors. These types of combined optimizations are often encountered when a real-world supply chain management is considered. However, although there is a large body of research on the classical version of the problem, a few articles only address combined optimization problems that seek to coordinate machine scheduling with delivering jobs in batches. Cheng et al. [20] consider a problem that arises when the objective is to minimize the sum of a function of number of batches and earliness penalties, where the earliness of a job is defined as difference between batch delivery time and the job completion time, but they have not considered the problem in presence of release times (all jobs are available at time 0). The complexity of some combined problems such as: makespan and completion times, when the jobs are to be delivered after their processing time to a customer or warehouse are measured by Lee and Chen [64]. Hurink and Knust [57] considered a flow shop problem for minimizing makespan with transportation times,

but they assume that a single robot which can shift only one job at a time does all transportation.

As cited in last chapters one of the problems identified by Hall and Potts is that of batching and sequencing on a single machine under the batch availability assumption in order to minimize sum of flow times plus delivery costs in presence of release time. Hall and Potts derive a forward dynamic programming algorithm for the problem under the assumptions of batch consistency.

A supplier's batch schedule and a manufacturer's batch schedule are batch consistent if for each pair of jobs i and j that are processed by the supplier and the manufacturer, whenever job i is in a strictly earlier batch than job j in the supplier's batch schedule, then job i is in an earlier batch or in the same batch as job j in the manufacturer's batch schedule [49]. They also make the further assumption of SPT-batch consistency, in which jobs with the same release times (i.e., jobs delivered in one batch by the supplier) and for the same customer are processed by the manufacturer in SPT order. Hall and Potts have proved that the overall time complexity of their algorithm for finding an optimal schedule is $O(n^{3H})$, where n and H identify the number of jobs and customers respectively.

In this chapter we consider the similar problem under the assumptions that for each pair of jobs i and j, whenever $p_i \leq p_j$ then $r_i \leq r_j$. This assumption may appear restrictive at first sight. However, within the framework of supply chain management this condition may be enforced as part of the coordination between the supplier (upstream stage) and the manufacturer.

In what follows, we study its structural properties, derive upper and lower bounds and offer a branch and bound scheme for solving this problem.

4.2 Problem Definition

Let $\{1,...,n\}$ denote the set of jobs to be scheduled on a single machine that can process at most one job at time. Each job is available at its release time and no preemption is allowed. Each job starts processing on the machine either at its release time if the machine is ready, or immediately after another job. Each job is produced for one customer and jobs are delivered to customers in batches. We denote the processing time and release time of job j by p_j and r_j respectively. A group of jobs

forms a batch if all are delivered to their respective customer at the same time. Two alternative batch formation strategies are allowed. The first is that of continuous batching, under which the jobs that form a batch are processed continuously; i.e., no job that belongs to another customer is processed between them and there is no idle time. The second strategy is that of discontinuous batching, where the jobs that form a batch are processed separately but delivered together, in which case at least one job that belongs to another customer is processed in between or there is idle time separating the two jobs. Let R_k denote the ready time of a continuous batch such that if processing of the first job of the batch is started at time R_k (or later), then all the others jobs of the batch can be processed continuously; i.e., at least the first job of the batch is ready at time R_k and other jobs are either already ready or become ready during the processing of earlier jobs. Let D_c denote the non-negative cost of delivering a batch to customer c. The objective function that we consider is to minimize the total flow time plus delivery cost. This is a natural extension of $1|r_j|\sum F_j$ problem to cater for coordination between the scheduling and distribution systems. The problem contains an additional term, which is delivery cost for each customer. Thus, using the standard classification scheme for scheduling problems [46], the objective function is $1|r_j|\sum F_j + \sum D_c k_c$, where k_c identifies the number of batches delivered to each customer. As described in the last section, the classical problem is NP-complete in the strong sense, even without the additional term that refers to batch delivery costs, and it is easy to show that recognition of problem is also NP-complete. However, by making some assumptions, we provide a branch and bound algorithm for solving the problem. These assumptions are that for each pair of jobs, $r_i \le r_j$ whenever $p_i \le p_j$ and that the jobs for each customer are to be processed in the shortest processing time (SPT) order.

4.3 Structural Properties

In this section, structural properties of the problem, used subsequently to derive upper and lower bounds, are analyzed. It is worth noting that since any result that is proved for completion time applies also to flow time, we state and prove the

following propositions and properties in terms of flow time, when in fact they are based on completion time considerations.

Proposition 4.3.1. *For a set of jobs, the sequence ordered by SPT is optimal in terms of total flow time.*

Proof (by contradiction): Consider a schedule formed from a sequence, *S, that* is not ordered by SPT. In this schedule there must be at least two adjacent jobs, say j followed by i, such that $p_i \leq p_j$ and $r_i \leq r_j$. Assume that the machine is ready for processing the two jobs at time t. Two situations need to be considered:

- $t \geq r_j$, in which case the total flow time of partial schedule composed of the two jobs concerned is

$$F_1 = (t + p_j) + (t + p_j + p_i) = 2t + 2p_j + p_i$$

- $t < r_j$ in which case

$$F_2 = (r_j + p_j) + (r_j + p_j + p_i) = 2r_j + 2p_j + p_i$$

Now perform adjacent pairwise interchange on jobs j and i to form a new sequence, S', with all other jobs remaining in their original positions. The completion times of all preceding jobs remain unchanged. However, the completion times of the two jobs interchanged and all succeeding jobs need to be considered.

- $t \geq r_j$ and certainly $t > r_i$. Therefore,

$$F_1' = (t + p_i) + (t + p_i + p_j) = 2t + 2p_i + p_j.$$

The completion time of jobs i and j combined does not change, nor as a result do the completion times of succeeding jobs. However, $F_1' < F_1$, and therefore total flow time decreases.

- $t < r_j$. Here one of four cases may arise:

  - $t < r_i$, $(r_i + p_i) < r_j$; in which case $F_2' = (r_i + p_i) + (r_j + p_j)$
  - $t < r_i$, $(r_i + p_i) \geq r_j$; in which case $F_2' = (r_i + p_i) + (r_i + p_i + p_j)$
  - $t \geq r_i$, $(t + p_i) < r_j$; in which case $F_2' = (t + p_i) + (r_j + p_j)$

- $t \geq r_i$, $(t + p_i) \geq r_j$;in which case $F_2' = (t + p_i) + (t + p_i + p_j)$

In each case, $F_2' < F_2$ and the completion time of jobs i and j combined is earlier. Hence, the completion times of all succeeding jobs are earlier. Due to both reasons, total flow time decreases.

Thus, interchanging the positions of j and i reduces overall flow time. Proceeding in this manner, carrying out any beneficial pairwise interchanges will ultimately yield a schedule based on a sequence ordered by SPT. □

The following corollary then follows immediately.

Corollary 4.3.2. *For a set of discontinuous batches with job release times, the sequence ordered by SPT is optimal in terms of total flow time.* □

Property 4.3.3 *When two jobs with processing times p_i and p_j and release times r_i and r_j form a continuous batch, b, with a total processing time P_b, ready for processing time, R_b, and machine ready time, t, one of the following four states occurs:*

- $r_i \geq t$ and $r_j \leq r_i + p_i$: $P_b = p_i + p_j$ and $R_b = r_i$.
- $r_i \geq t$ and $r_j > r_i + p_i$: $P_b = p_i + p_j$ and $R_b = r_j - p_i$.
- $r_i < t$ and $r_j \leq t + p_i$: $P_b = p_i + p_j$ and $R_b = t$.
- $r_i < t$ and $r_j > t + p_i$: $P_b = p_i + p_j$ and $R_b = r_j - p_i$.

The above property is easily extendable to more than two jobs.

Proposition 4.3.4. *For a set of continuous batches with constrains on the ready times, R, such that whenever $T_i < T_j$ then $R_i < R_j$, where T is the batch effective time defined as $T_b = \frac{A_b}{\delta_b}$, with A_b being the total processing time of the batch and δ_b the batch size (number of jobs in the batch), which could be equal to 1, the sequence*

ordered by the Shortest Effective Batch Time (SEBT)is optimal in terms of total flow time.

Proof: (by contradiction). Consider that a schedule S, which is not ordered by the SEBT rule, is optimal. In this schedule, there must be at least two adjacent batches, say Y and X such that Y is followed by X when $R_X < R_Y$ and $T_X < T_Y$.

Assume that the machine is ready for processing the two batches at time t. Two situations need to be considered:

- $t \geq R_Y$, in which case the total flow time of the partial schedule composed of the two batches concerned is

$$F_1 = (t + A_Y)\delta_Y + (t + A_Y + A_X)\delta_X$$

- $t < R_Y$, in which case

$$F_2 = (R_Y + A_Y)\delta_Y + (R_Y + A_Y + A_X)\,\delta_X$$

Now perform adjacent pairwise interchange on batches Y and X to form a new sequence, S', with all other batches remaining in their original positions. The completion times of all preceding batches remain unchanged. However, the completion times of the two batches interchanged and all succeeding batches need to be considered.

- $t \geq R_Y$ and certainly $t \geq R_X$. Therefore,

$$F_1' = (t + A_X)\delta_X + (t + A_X + A_Y)\delta_Y .$$

The completion time of batches X and Y combined does not change, nor as a result do the completion times of succeeding batches. However $F_1' < F_1$ and therefore, total flow time decreases.

- $t < R_Y$. Here one of four cases may arise:
  - $t < R_X$, $(R_X + A_X) < R_Y$; in which case $F_2' = (R_X + A_X)\delta_X + (R_Y + A_Y)\delta_Y$
  - $t < R_X$, $(R_X + A_X) \geq R_Y$; in which case $F_2' = (R_X + A_X)\delta_X + (R_X + A_X + A_Y)\delta_Y$

- $t \geq R_X$, $(t + A_X) < R_Y$; in which case $F_2' = (t + A_X)\delta_X + (R_Y + A_Y)\delta_Y$
- $t \geq R_X$, $(t + A_X) \geq R_Y$; in which case $F_2' = (t + A_X)\delta_X +$ $(t + A_X + A_Y)\delta_Y$

In each case, $F_2' < F_2$ and the completion time of batches X and Y combined is earlier. Hence, the completion times of all succeeding batches are earlier. Due to both reasons, total flow time decreases.

Thus, interchanging the positions of X and Y reduces overall flow time. Proceeding in this manner, carrying out any beneficial pairwise interchanges will ultimately yield a schedule based on a sequence ordered by SEBT. □

Proposition 4.3.5. *For a set of continuous batches, in the absence of any constraints on the inter-relationships among job release times, if preempt-resume is allowed then the sequence ordered by the Shortest Effective Remaining Batch Time (SERBT) is optimal in terms of total flow time, with batch effective time being* $T = \frac{A_b}{\delta_b}$.

Proof: Generalizing the idea from Baker [8] for minimizing the sum of flow times for a set of jobs and in light of proposition 4.3.4, we can show that when preempt-resume prevails the optimal rule for a set of batches is to always keep the machine assigned to the available batch with minimum remaining Effective Batch time. □

Proposition 4.3.6. *Assume a partial schedule, where some batches have been formed on each machine, but no decision has yet been taken on batching the remaining 'un-batched' jobs. Here, a lower bound on the sum of job flow times of an optimally completed schedule corresponds to the sum of job flow times in a schedule formed by considering each un-batched job as a single-job batch and sequencing all batches in the order of SEBT or SERBT.*

Proof: When completing the schedule, any batching of un-batched jobs will necessarily delay some jobs. Hence, considering each such job as a single-job batch ensures no delay. Thereafter, SEBT sequencing for the case that ready times, R , satisfies the constraint cited in proposition 4.3.4 ensures that the resulting schedule minimizes total flow time by virtue of proposition 4.3.4. Otherwise SEBRT

sequencing ensures that the resulting schedule minimizes total flow time by virtue of proposition 4.3.5. □

In addition to the propositions, corollary and property that are proved in above, some of the propositions and corollaries cited in the last chapter have also dominance on the structure of this problem. These propositions are modified and rewritten in the following.

Proposition 4.3.7. *In an optimal solution, any batch destined for a customer j that has* $\delta_b > 1$ *jobs will have the property that* $(\delta_b - 1)p_l < D_c$, *where l is the last job in the batch.*

Proof: See the proof of proposition 3.3 in chapter 3. □

The following corollary then follows immediately.

Corollary 4.3.8. *In an optimal solution, any job that has a processing time greater than the batch delivery cost to the corresponding customer, i.e., such that* $p_i > D_c$, *will form a single-job batch.*

Proposition 4.3.9. *If batches belonging to the same customer are concatenated in the same order with which they occur in the optimal schedule, then the jobs will appear ordered by SPT.*

Proof: See the proof of proposition 3.5 in chapter 3. □

Proposition 4.3.10. *A lower bound on the number of batches destined for a customer in an optimal solution can be found by the following greedy maximum batching algorithm: take the jobs destined for the customer concerned in SPT order; if the job may be added to the current batch by virtue of proposition 4.3.7, then add it; else start a new batch.*

Proof: See the proof of proposition 3.6 in chapter 3. □

Proposition 4.3.11. *Let a job k be in position* s_r *to the right of a batch b, which is in position* s_l *in a SEBT sequence (in which job k constitutes a single-job batch). If k may be added to b by virtue of proposition 4.3.7, then the change in the sum of job*

flow times resulting from doing so could be found by updating the contribution of the batches between s_l and s_r inclusive.

Proof: See the proof of proposition 3.7 in chapter 3. □

Proposition 4.3.12. *In completing a partial schedule, where some batches have been formed, but no decision has been taken yet on batching the remaining `un-batched' jobs, a 'batching penalty' Δ_k, attaches to each un-batched job, since if the job is added to the last formed batch for the customer concerned, total flow time will increase, and if a new batch is started with it, an additional batch delivery cost will be incurred. Moreover, $\delta_l p_k \leq \Delta_k \leq D_c$, where δ_l is the number of jobs in the batch that it may join and D_c is the appropriate batch delivery cost.*

Proof: See the proof of proposition 3.8 in chapter 3. □

Subsequently, we will use proposition 4.3.5, which is based on preempt-resume. Using preempt-resume for finding lower bound ensurs that if the lower bound is still less than the upper bound we are allowed to prune the branch, but on the other hand, since the lower bound is not bonded perfectly and furthermore, we are not allowed to use preempt-resume, the value that we will find at leaf node must be considered again without using proposition 4.3.5. The following proposition will help to reduce the search space area for the given sequence at leaf nodes.

Proposition 4.3.13 *For a set of batches that have been established under the conditions defined in section 2, the optimum sequence in terms of total flow time is a sequence that if sorted in SEBT order gives the release dates of batches as:*

A) Either sorted in non- decreasing order,

B) Or if for the partial schedule of the sequence the order of the release dates conflict with the order of the batches, i.e. the batches are ordered in SEBT but the release dates are not in non-decreasing order, then for the given partial schedule we can claim that $R_x + P_x \geq R_y$, while R and P show the ready time and total processing time of batches and x and y identify the index of any given batch in the partial schedule being considered.

Proof: See appendix 1. □

4.4 Branch and Bound Scheme

Search of the solution space is structured as a trivalent 0-1-2 search tree, where each node is partitioned into three: one indicating that a job is added to the last batch for the customer concerned continuously (1), another indicating that a job is added to the last batch for the customer concerned discontinuously (2), and a third indicating the start of a new batch that could be a single-job batch (0). The tree is constructed in a depth-first fashion. At the beginning, all jobs that have to form single-job batches by virtue of corollary 4.3.8 are identified and their corresponding variables are set to 0 once for all. Other components of the branch and bound scheme are presented in the following subsections.

4.4.1 Branching and ordering of variables

Variables are ordered in accordance with the SPT of the corresponding jobs. At each node of the decision tree, two tasks are performed. First, variables that have to be set to zero, because no batch for the customer concerned has been formed or by virtue of proposition 4.3.7, are set to zero. Secondly, the first free variable (free variables are the ones that have not yet been committed to either zero or one) in the SPT sequence is set to one.

4.4.2 Fathoming and backtracking

A node is fathomed if:

Either, it is a leaf node, i.e., all variables are fixed.

Or, the lower bound exceeds or equals the incumbent upper bound.

Fathoming initiates backtracking to the first node associated with a variable whose value is either 1 or 2. If the value of this variable is 1 then it is set to 2; else it is set to zero. If no such node is found, the search terminates.

4.4.3 Upper bounds

As cited in the last chapter, providing a sharp, i.e., low, initial upper bound is critically important for enhancing the exclusion rate of the branch and bound algorithm, i.e., the rate with which nodes are fathomed. Hence, it is worth expending some computational effort to achieve that end. It is worth noting that finding a good

heuristic upper bound for the problem in presence of release times is more difficult than the problem in which all jobs are ready at time zero. However, two algorithms are provided, the first one finds an initial upper bound and the second one improves it.

Algorithm *UB*1—initial upper bound

Begin

- Form all jobs into single-job batches and sequence the batches in SEBT order (which, in this case, is equivalent to SPT), and call it original sequence.

For i =1 to number of batches **do**

- Select the first single-job batch in the sequence, which is not selected yet.

For $j = i + 1$ to number of batches **do**

- Scan forward until the first single-job batch with the same customer to the selected batch, i.e. batch i, is found and call it k.

- Move the newly found single-job batch, i.e. batch k, to the position of batch i, and join them to make a bigger batch, if they will join profitably.

- Continue the interior loop until no improving move is found.

- Continue the exterior loop until a complete scan of all batches.

end.

Algorithm *UB*2—improving upper bound

Moving a job is defined as either moving it to the preceding batch for the same customer if it is the first job in the batch under consideration or to the succeeding batch if it is the last job in the batch under consideration.

Begin

Start with a given schedule that is found by algorithm *UB*1 and is sequenced in SEBT.

Repeat

For i =1 to number of batches **do**

Take 1st job of batch. If it has not been moved in the current iteration and it can be moved profitably, move it and adjust the schedule.

Take last job of batch. If it has not been moved in the current iteration and it can be moved profitably, move it.

until no improving move is found.

Repeat

Find a job that can be moved profitably. Move it and adjust schedule.

until; no improving move is found.

end.

4.4.4 Lower bounds

The structure of lower bound on total flow time and batch delivery costs at each node is the same as implemented for the last problem in chapter 3. However, for clarification we adopt and rewrite it as follows.

It is worth recalling that at each node of the decision tree, if in view of the batching decisions already taken, a job has to start a new batch, then the partial solution is immediately augmented by a batch starting with that job.

At each node of the decision tree, a lower bound on total flow time is calculated in accordance with proposition 4.3.6. It should be noted that proposition 4.3.6 implies that if the sequence which is sorted in SEBT satisfies the constraint cited in proposition 4.3.4, i.e. whenever $T_i < T_j$ then $R_i < R_j$, then lower bound on total flow time will be calculated by virtue of this proposition, otherwise, lower bound on total flow time will be calculated by virtue of proposition 4.3.5. Additionally, batch delivery costs are added for each batch already formed.

Furthermore, a lower bound on the batching penalties is calculated by applying the logic of proposition 4.3.12 in the following way. The first un-batched job of a customer, k, will either start a new batch or join the last batch. It would, therefore, attract a lower bound on its batching penalty $= \delta_l p_k$, where δ_l is the number of jobs in the last batch of the customer concerned. Since for each subsequent job, we do not know the number of jobs in the batch that it may join in the optimal completion of the current partial solution, each such job would attract a lower bound on its batching

penalty = p_k i.e. the lowest batching penalty that it may incur would be if it joined a single-job batch.

At the initial stages of the search, opportunities arise for tightening the overall lower bound still further by considering proposition 4.3.10. Assume that the number of batches already formed for a particular customer in the current partial solution is b_k and that the minimum number of batches is $b_{\min}$. It would then be possible to raise the sum of lower bounds on batching penalties by identifying the $b_{\min} - b_k$ highest individual lower bounds on batching penalties and replacing each by the batch delivery cost.

The overall lower bound is then the sum of the lower bound on the total flow time, batch delivery costs of the batches already formed and the sum of the lower bounds on the batching penalties for the un-batched jobs.

4.4.4.1 Optimum value at Leaf nodes

Since we are not allowed to use preempt-resume, the value that we find at leaf node must be considered again without using proposition 4.3.5 as follows.

- If the obtained value for the lower bound at leaf nodes is calculated without using preempt-resume and this value is less than upper bound then, this value is certainly an optimum or at least a local optimum. It is worth noting that it is immaterial if we have used preempt-resume at some earlier stages.

- If the obtained value for the lower bound at leaf nodes is calculated with using preempt-resume, then we have to consider all possible options that may lead us to the minimum value. For achieving this objective the sequence has to be ordered in SEBT and the partial schedules in which the total processing time and ready time of batches conflict together must be recognized. The total processing time and ready time of batches in a partial schedule conflict together when the batches are ordered in SEBT but the release times are not in non-decreasing order.

  We start the process of a given sequence from the first batch in the sequence and whenever we faced a partial schedule in which the batches conflict together, consider all possible options and select the best one. It is worth

noting since the jobs for each customer should be processed in SPT order, and also in the light of proposition 4.3.13, the number of extra states that should be considered for each partial schedule is less than or equal to the number of customers. In fact, proposition 4.3.13 ensures that for a given partial schedule, when the processing of a batch is completed, all of the remaining batches of the partial schedule will be ready for processing. In other words, it means that the remaining batches satisfy the constraints cited in proposition 4.3.4.

4.4.5 Numerical example

Consider the following two-customer problem, with delivery costs of 200 and 100 respectively:

	Customer 1			Customer 2	
	Job 1	Job 2	Job 3	Job 1	Job 2
Processing time	10	65	160	20	60
Release date	0	50	60	15	40

Let (ij) denote the i th job destined for the j th manufacturer.

Initial lower bound:

The lower bound on total flow time, *LBF*, corresponding to flow time under SPT = 635. Batch delivery costs, *LBD*, is 300 (we have to start a batch for each customer). The lower bound on batching penalties, *LBB* is: (65+200) + (60) = 325, where the numbers within the first (second) bracket correspond to the penalties incurred by each job of the first (second) customer in turn. Note that the 200 for the first customer results from substituting the initial lower bound on the batching penalty, which is equal to 160, by the batch delivery cost, by virtue of the fact that the minimum number of batches for the first customer is two.

$LB = LBF + LBD + LBB = 635 + 300 + 325 = 1260$

Initial upper bound:

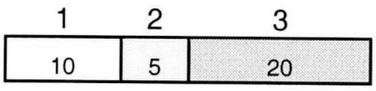
Applying algorithm *UB*1 to the 5 single-job batches leads to the following schedule: |(11)|, |(12) (22) |, |(21)(31)|, with a total flow time of 860. Adding total batch delivery costs gives UB* = 860 + 500 = 1360.

Applying algorithm *UB2* for this schedule cannot improve the upper bound, therefore, UB* = 1360.

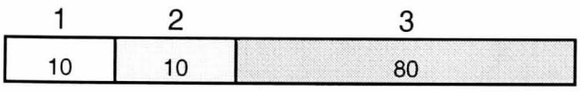
Branch and Bound:

Let symbol { } shows a discontinuous batch. In what follows the information appropriate with each node is presented. Furthermore, the starting and completion times of batches, which we have taken decision about, are illustrated. The remaining single-job batches, which we have not decided about yet, must be sequenced in SEBT at the end of the partial schedule. The numbers, at the top of each operation, show the order of operations (assuming that idle time of machine is also an operation). The operations that may confuse the reader are explained separately.

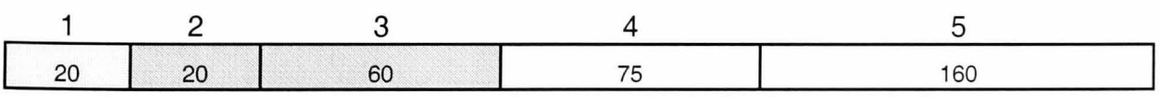
- S_0 = |(11) |, | (12)|; LBF = 635, LBD = 300, LBB = 65 + 200 + 60 = 325, LB = 1260.



- S_1 = |(11)|, | (12) (22) |; LBF = 700, LBD = 300, LBB = 65 + 200 = 265, and LB = 1265.



- S_2 = |(12) (22)|, | (11) (21) |, |(31)|; LBF = 885, LBD = 500, LBB = 0, and LB = 1385. Since LB > UB*, backtrack. Notice that in this node we have used preempt-resume.



Explanation:

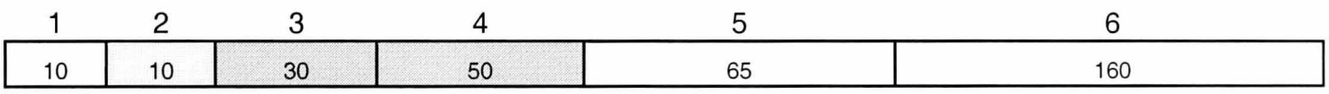
1: The time we have to wait till the first batch gets ready. It is worth noting that at time 0, job (11) is ready but it cannot be processed because at this node job (11) is one part of batch b_1 = | (11) (21)| which is not ready yet.

2: The first batch in the sequence, when the sequence is ordered in SEBT, is batch b_1. Since this batch is not ready at time 20, we use preempt-resume and then the process starts with batch b_2 = |(12) (22)|.

3: The process must continue until the next batch gets ready (arrives). Next batch, b_1 is ready at time 40. At time 40, the total processing time of batch b_1, is 75, while the remaining total processing time of batch b_2 is 60. Since the size of both batches is the same, the process must continue with batch b_2.

Steps 4 and 5 are clear.

- S_3 = {| (11) (21)|}, |(12) (22)|, | (31)|; LBF = 855, LBD = 500, LBB =0, and LB = 1355. In fact, this is the best solution, which is found by now. Therefore UB* = 1355, backtrack



Explanation:

1: At time 0, the process of the first job of discontinuous-batch b_1' = {| (11) (21)|}, with the processing time of 10, starts and will be completed at time 10. It is worth noting that in contrast with node S_2, since batch b_1' is a discontinuous-batch we should not wait till the whole batch gets ready.

2: The idle time of machine till the next batch (job) gets ready.

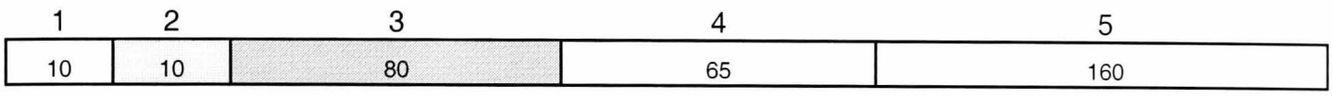
3: The second job of batch b_1' is ready at time 50, while batch b_2 = |(12) (22)| is ready at time 20. Since we are using preempt-resume the process could continue with batch b_2 until the second job of batch b_1' gets ready.

4: At time 50 the second job of batch b_1' is ready. At time 50, the remaining total processing time of batch b_1' is 65, while the remaining total processing time of batch b_2 is 50. Since the size of both batches is the same, the process must continue with batch b_2.

5: The process of the second job of batch b_1' starts at time 100 and will be completed at time 165.

6: The process of single-job batch |(31)| starts at time 165 and will be completed at time 325.

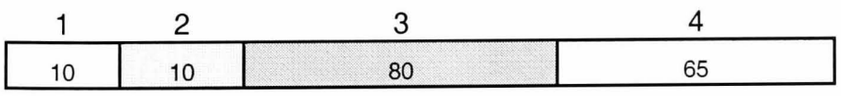
Since we are at leaf node and the result has obtained by using preempt-resume, two possible options, without using preempt-resume, must be evaluated. The first option occurs when batch |(12) (22)| is followed by the second job of discontinuous-batch {| (11) (21)|}; in which case LBF = 855, LBD = 500, LBB =0, and LB = 1355.



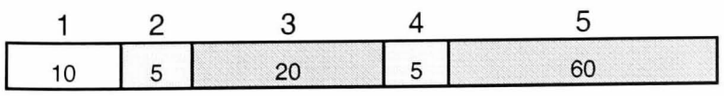
The second option occurs when the second job of discontinuous-batch {|(11) (21)|} is followed by batch |(12) (22)|; in which case LBF = 975, LBD = 500, LBB =0, and LB = 1475.

The LB for the first option is still less than the current upper bound, thus it is the best solution, which is found by now. Therefore UB* = 1355, backtrack.

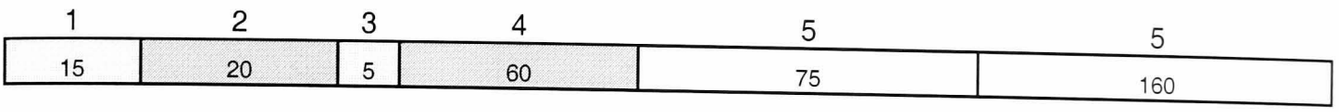
- S_4 = |(11)|, | (12) (22)|, |(21)|; LBF = 700, LBD = 500, LBB = 160, and LB = 1360. Since LB > UB*, backtrack.



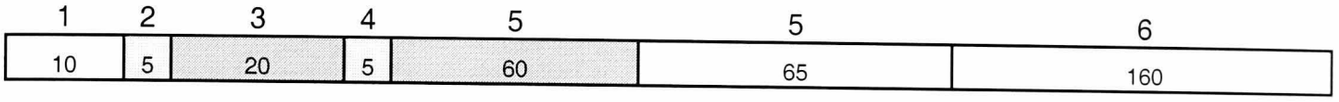
- S_5 = |(11)|, {|(12) (22)|}; LBF = 700, LBD = 300, LBB = 65 +200 = 265, and LB = 1265.



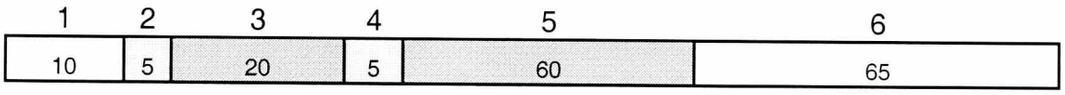
- S_6 = |(11)(21)|, {|(12) (22)|}, |(31)|; LBF = 885, LBD = 500, LBB = 0, and LB = 1385. Since LB > UB*, backtrack. Notice that in this node we don't need to use preempt-resume.



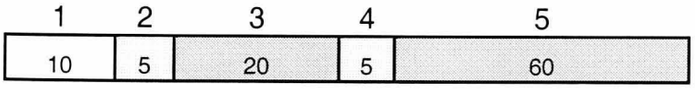
- S_7 = {|(11)(21)|}, {|(12) (22)|}, |(31)|; LBF = 855, LBD = 500, LBB = 0, and LB = 1355. Since LB = UB*, backtrack.



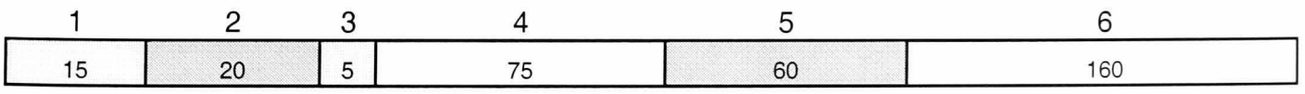
- S_8 = |(11)|, {|(12)(22)|}, |(21)|; LBF = 700, LBD = 500, LBB = 160, and LB = 1360. Since LB > UB*, backtrack.



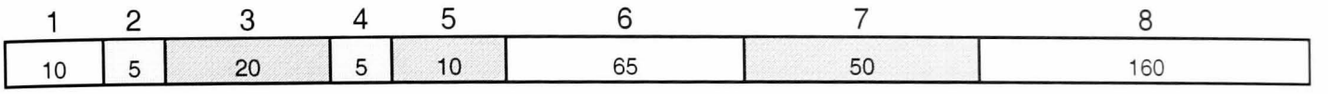
- S_9 = |(11)|, |(12)|, |(22)|; LBF = 635, LBD = 400, LBB = 65 +200 = 265, and LB = 1300.



- S_{10} = |(12)|, |(11)(21)|, |(22)|, |(13)|; LBF = 775, LBD = 600, LBB = 0, and LB = 1375. Since LB > UB*, backtrack.



- S_{11} = {|(11)(21)|}, |(12)|, |(22)|,|(13)|; LBF = 755, LBD = 600, LBB = 0, and LB = 1355. Since LB = UB*, backtrack. Notice that in this node we use preempt-resume.



Explanation:

1: At time 0, the process of the first job of discontinuous-batch $b_1' = \{|(11)(21)|\}$, with the processing time of 10, starts and will be completed at time 10. It is worth noting that since batch b_1' is a discontinuous-batch we should not wait until the whole batch gets ready.

2: The idle time of machine till the next batch (job) gets ready.

3: At time 15, the single-job batch |(12)|, with the processing time of 20, starts and will be completed at time 35.

4: At time 35 no batch (jobs) is ready, then the machine will be idled till the next batch (job) gets ready.

5: The next batch (job) in the sequence, when the sequence is ordered in SEBT, is the second job of batch b_1', job |(21)|. Since this job is not ready at time 40, we use preempt-resume and the process continue with job |(22)| till job |(21)| gets ready.

6: At time 50, job |(21)| is ready. The processing time of this job is 65 and it belongs to the discontinuous-batch b_1' with size of 2. On the other hand, the remaining total processing time of single-job batch |(22)|, i.e. the job that is under processing, is 50 and its size is 1. Therefore, by virtue of proposition 4.3.5 the processing of job |(22)| must be interrupted and the processing of job |(21)| starts.

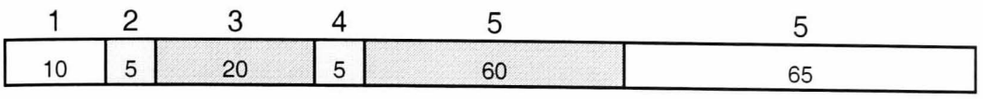
7: At time 115, the processing of job |(21)| is completed and the process of job |(22)|, with the remaining processing time of 50, starts.

8: The process of single-job batch |(31)| starts at time 165 and will be completed at time 325.

It is worth noting that the total flow time at this node must be calculated as follows.

$$\text{LBF} = (15 + 20) + (50 + 65) \times 2 + (115 + 50) + (165 + 160) = 755.$$

- S_{12} = |(11)|, |(12)|, |(22)|, |(21)|; LBF = 635, LBD = 600, LBB = 160, and LB = 1395.



- Search Completed.

4.5 Computational Results

In the absence of benchmark test problem instances upon which to test the branch and bound algorithm (B&B) and compare its performance with that of the only other available algorithm for the single machine-scheduling problem under consideration; i.e., the dynamic programming algorithm (DP) of Hall and Potts, we resorted to generating a set of problem instances.

For the set, three subsets were generated; one with 2 customers, another with 3 and the third with 4. In each subset, the number of jobs was varied up to a total of 40. The jobs were randomly distributed among customers, with each being assigned at least two jobs. Processing times were randomly generated integers from the uniform distribution defined on [1,100]. To ensure that results are representative, 5 instances were generated for each combination of number of jobs-number of customers. Each of the running times in tables below represents the average over the appropriate five instances.

Since interaction between batch delivery costs and job processing times may affect problem hardness, we generated two classes of problems. In class A, all job processing times are less than the delivery cost of the relevant customer, while in class B it is possible randomly that some jobs have a processing time that exceeds the delivery cost.

The computational experiments were run on a Pentium 4 computer with 2.40 GHz of CPU and 512 MB of RAM. Both B&B and DP were coded in C++.

Comparative results are shown in tables 1 to 6. Furthermore, to present the efficiency of B&B, another subset with 8 customers were also generated. Tables 7 and 8, show the results for this subset.

As can be clearly seen, the DP algorithm has advantage over B&B only for some class A problem instances having 2 customers. For all other instances, B&B is more efficient by far. This is due to the time complexity of the DP algorithm, which can be clearly observed in the escalation of computing time with the increase in the number of jobs and customers. On the other hand, the efficiency of the B&B, which enables it to solve problem instances very fast, is attributable to the effectiveness of the initial upper bound and the sharpness of the lower bounds. Careful analysis of the structural properties of the problem proved crucial in forging both sub-algorithms (upper bound and lower bound).

As cited in the last chapter the complexity of the dynamic programming algorithm is such that for the same number of jobs, problem instances with a larger number of customers are more difficult while, the opposite is true for the branch-and-bound algorithm, as revealed by the solution times. This is so because when the number of customers is large, the algorithm is in effect searching over a larger number of smaller subsequences, with much smaller numbers of possible permutations. However, it is expected that by increasing the number of jobs for the same number of customers the running times increase. In this regard, we define a critical point (CP), which shows the number of jobs for which the B&B algorithm can still solve problem instances efficiently. It is worth noting that CP is not the maximum number of jobs that can be solved by B&B, but only the number of jobs after which the running time increases critically. The CP for each combination of customer-jobs is shown in the caption of diagrams 1 to 8.

Table 1: Running times for problem instances with 2 customers, where jobs randomly distributed among customers-Class *A*.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
2	7	36	16
2	12	232.2	26
2	17	1594.2	38.2
2	22	6860	212.2
2	27	18837.2	3174.6
2	32	52944.2	56645.4
2	37	124341	1038527.4
2	40	213843	27518645

Diagram 1: Running times versus Number of jobs for problem instances with 2 customers, where jobs randomly distributed among customers-Class *A*. CP = 37.

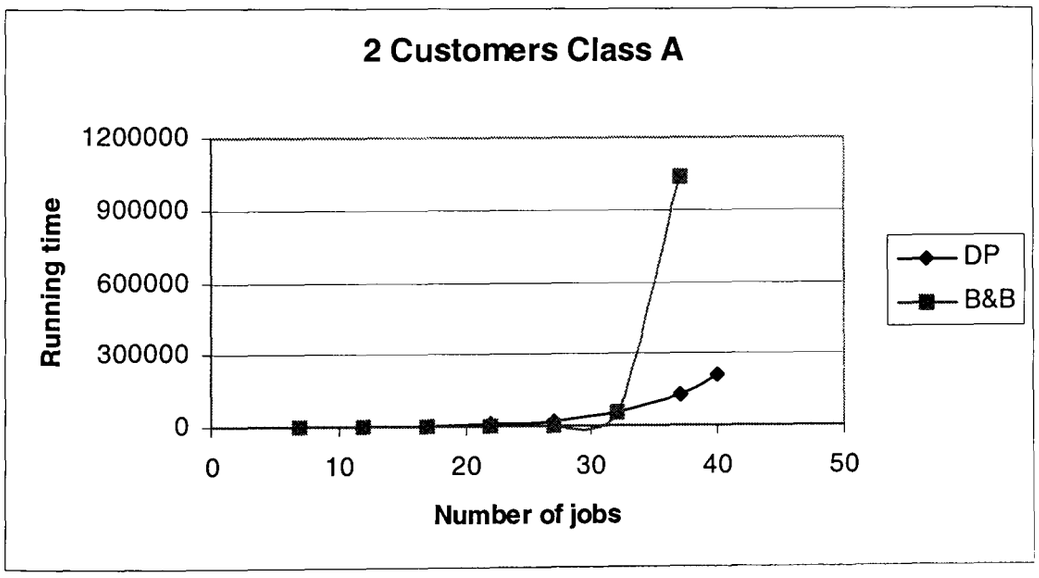


Table 2: Running times for problem instances with 3 customers, where jobs randomly distributed among customers-Class *A*. *: Computer gives up for lack of sufficient.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
3	7	120	14
3	12	3166.6	16
3	17	28641	28
3	20	137934.4	52
3	22	280443.34	170.4
3	27	937307.8	1948.8
3	32	*	4873
3	37	*	99825.6
3	40	*	213447

Diagram 2: Running times versus Number of jobs for problem instances with 3 customers, where jobs randomly distributed among customers-Class *A*. CP = 42.

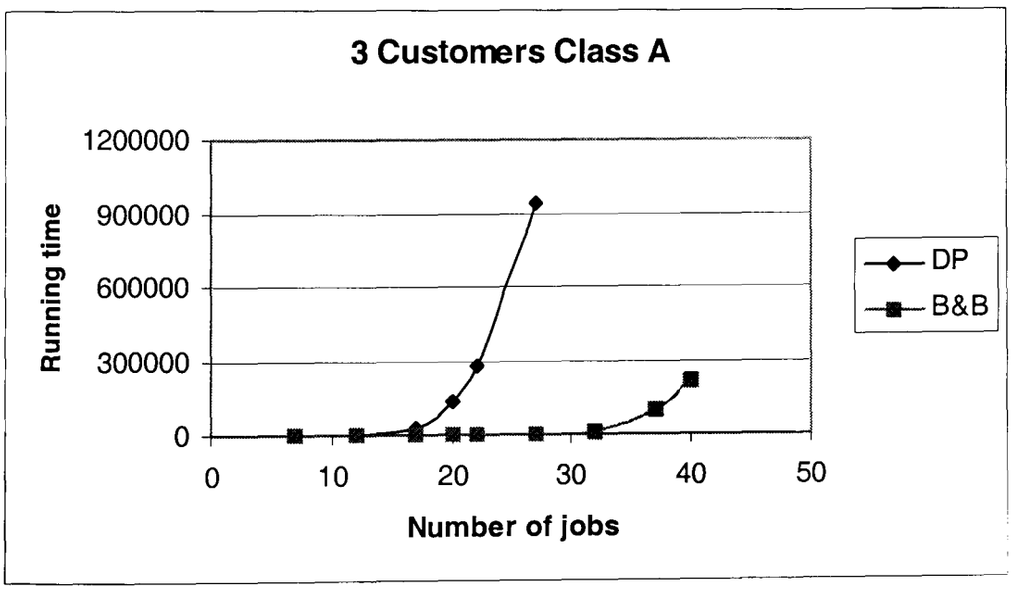


Table 3: Running times for problem instances with 4 customers, where jobs randomly distributed among customers-Class *A*. *: Computer gives up for lack of sufficient.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
4	8	889.4	20
4	10	4117.8	20
4	13	39174.6	20
4	15	131477	22
4	16	586020.4	26
4	18	4221419.5	40.2
4	23	*	134.2
4	28	*	843.2
4	32	*	4350.2
4	37	*	45717.6
4	40	*	80263.4

Diagram 3: Running times versus Number of jobs for problem instances with 4 customers, where jobs randomly distributed among customers-Class *A*. CP = 48.

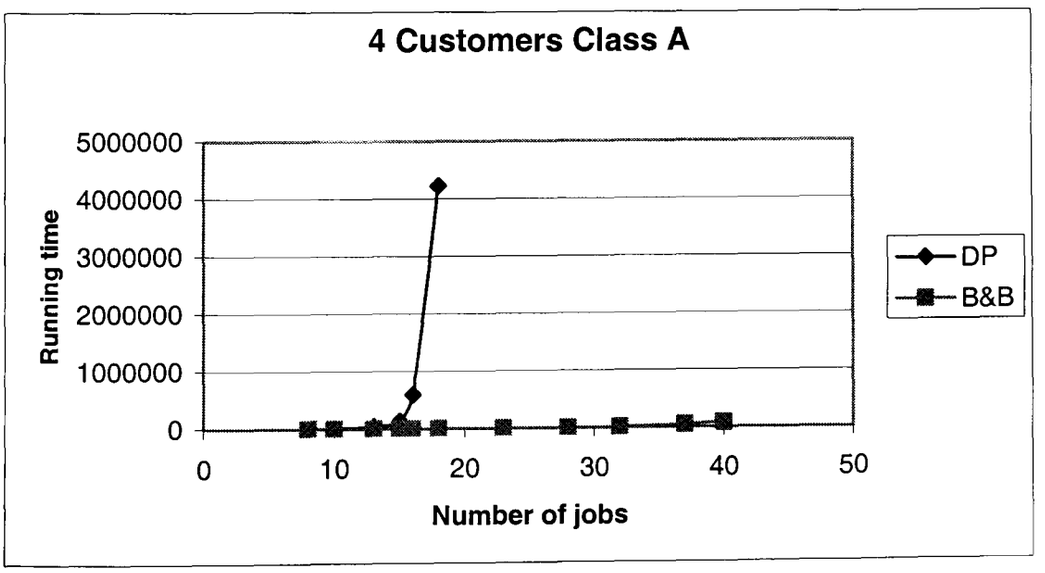


Table 4: Running times for problem instances with 2 customers, where jobs randomly distributed among customers-Class *B*.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
2	7	94.2	14
2	12	288.4	14
2	17	1686.4	16
2	22	6653.6	22
2	27	19768.6	102.2
2	32	54968.8	102.2
2	37	126511.6	19592.2
2	40	207202.2	21833.75

Diagram 4: Running times versus Number of jobs for problem instances with 2 customers, where jobs randomly distributed among customers-Class *B*. CP = 48

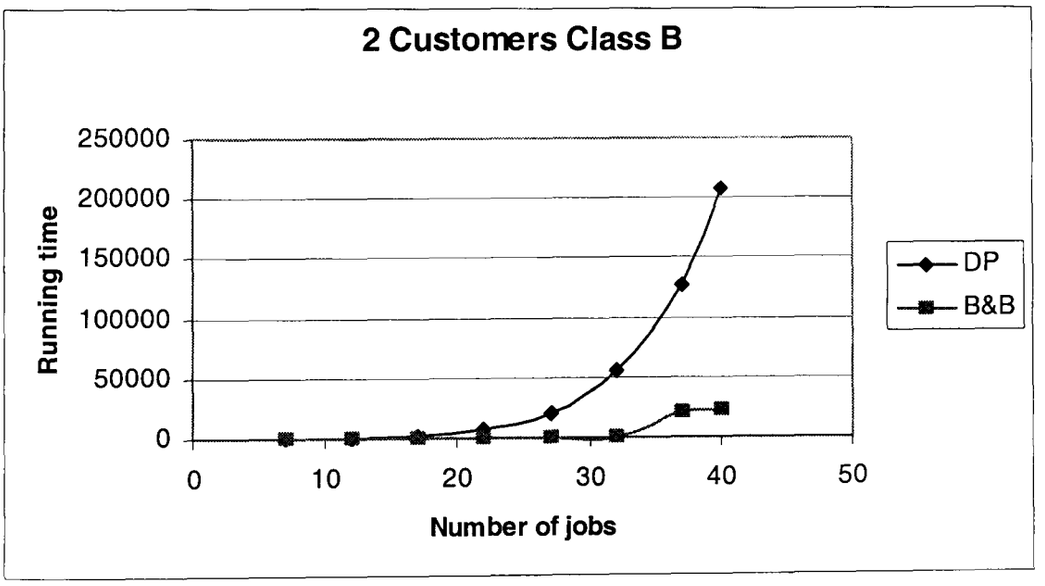


Table 5: Running times for problem instances with 3 customers, where jobs randomly distributed among customers-Class *B*. *: Computer gives up for lack of sufficient.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
3	7	120	12
3	12	2475.4	16.2
3	17	35260.6	14
3	20	125089.6	34
3	22	798830.6	32
3	27	*	34.2
3	32	*	10
3	37	*	32
3	40	*	358

Diagram 5: Running times versus Number of jobs for problem instances with 3 customers, where jobs randomly distributed among customers-Class *B*. CP =58.

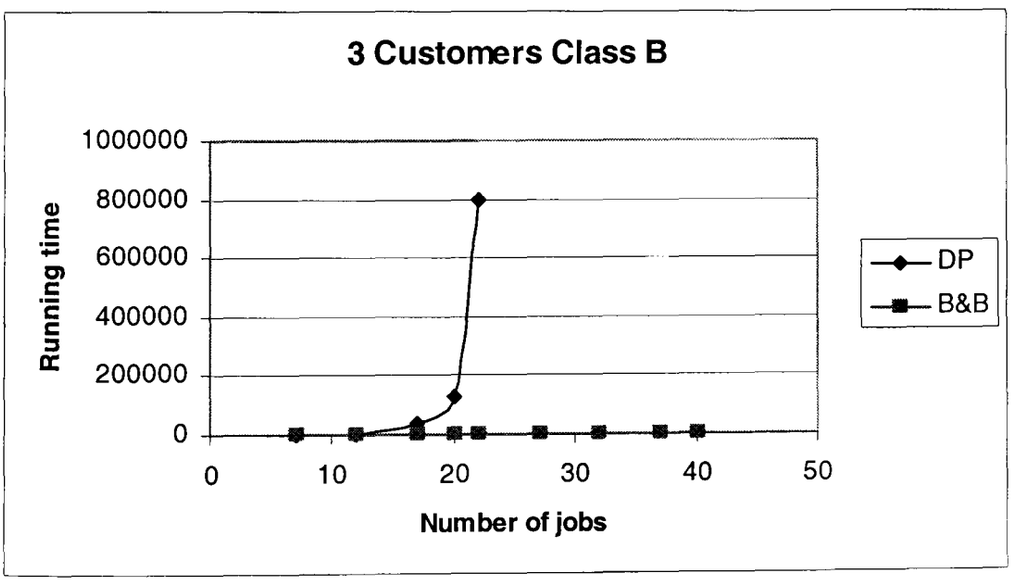


Table 6: Running times for problem instances with 4 customers, where jobs randomly distributed among customers-Class *B*. *: Computer gives up for lack of sufficient.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
4	8	863.2	18
4	10	4439	12
4	13	40209.8	14
4	16	364484.2	14.2
4	18	4287873.8	20
4	23	*	20
4	28	*	22
4	32	*	70.2
4	37	*	446.6
4	40	*	779.2

Diagram 6: Running times versus Number of jobs for problem instances with 4 customers, where jobs randomly distributed among customers-Class B. CP = 63.

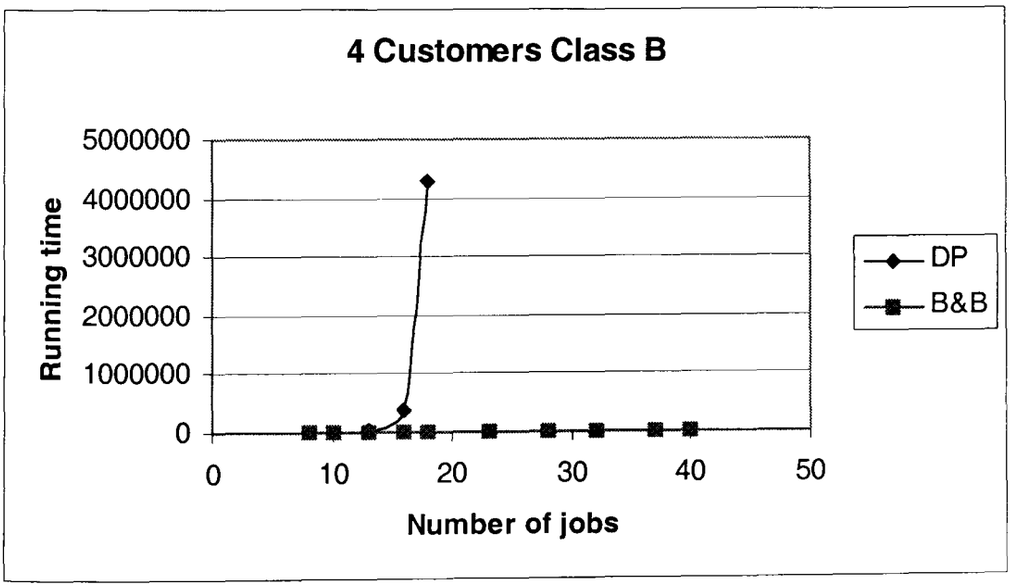


Table 7: Running times for problem instances with 8 customers, where jobs randomly distributed among customers-Class A.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
8	16	_	24.2
8	21	_	20
8	26	_	42
8	31	_	294.4
8	36	_	979.4
8	40	_	2503.8

Diagram 7: Running times versus Number of jobs for problem instances with 8 customers, where jobs randomly distributed among customers-Class A. CP = 52.

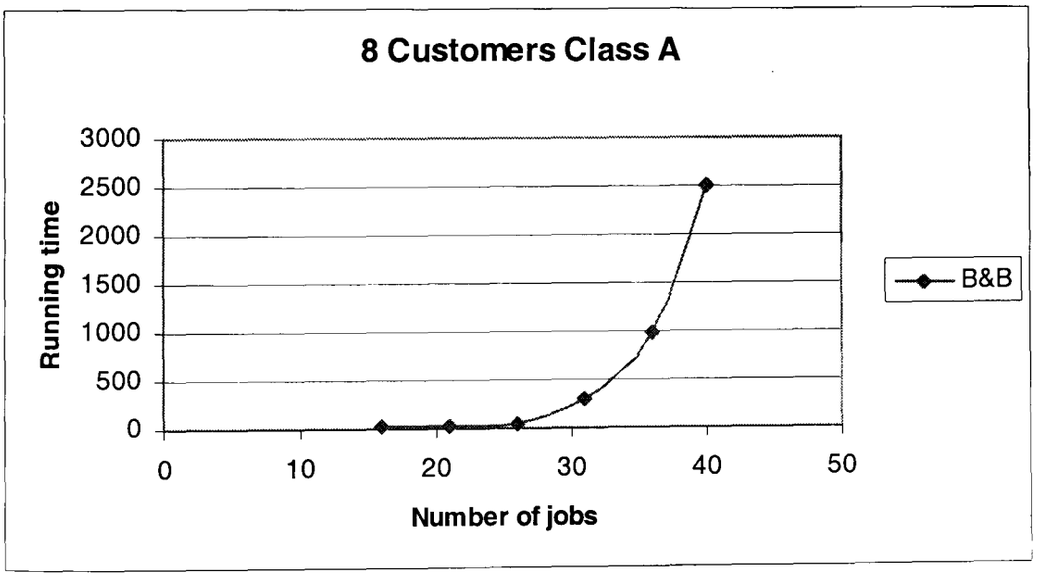


Table 8: Running times for problem instances with 8 customers, where jobs randomly distributed among customers-Class B.

Number of Customers	Number of jobs	Running time (per ms)	
		DP	B&B
8	16	–	20
8	21	–	22
8	26	–	32.2
8	31	–	50
8	36	–	58.2
8	40	–	236.2
8	45	–	464.8

Diagram 8: Running times versus Number of jobs for problem instances with 8 customers, where jobs randomly distributed among customers-Class B. CP = 70.

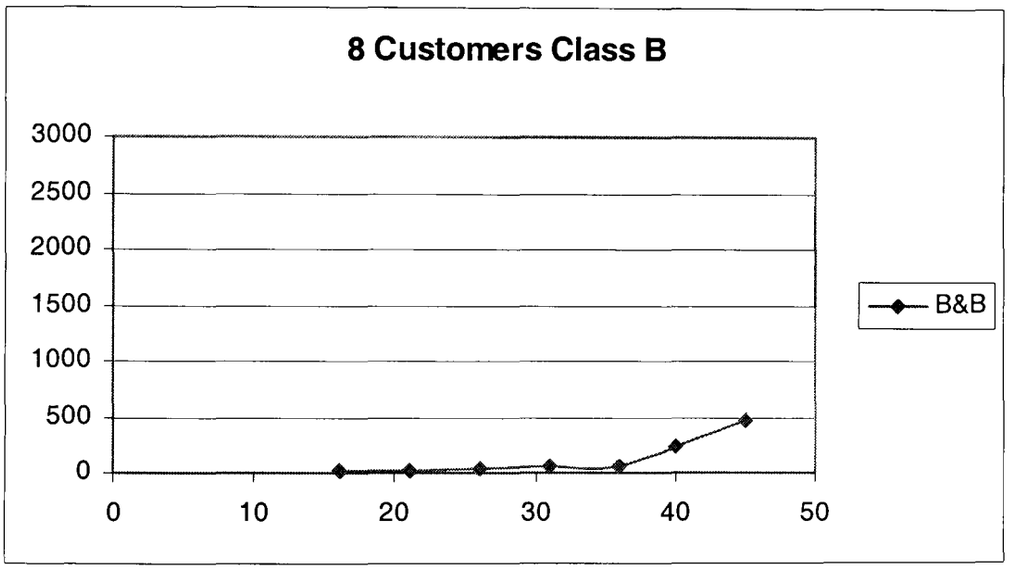


Table 9: Average percentage Relative Error.

	Random Distribution of Jobs	
Customers	Class A	Class B
2	0.28	0.07
3	0.33	0.08
4	0.19	0.08
8	0.2	0.07

Effectiveness of the upper bound makes it possible to use it as a fast heuristic. Table 9 shows that on average it produces solutions that are within 0.33% of the optimum (relative error = $(\frac{UB}{Optimal} - 1)$). Moreover, the error is smaller for larger instances. However, it may be argued that the efficiency of the B&B obviates the need for a heuristic solution.

4.6 Conclusion

A branch and bound algorithm for scheduling a set of jobs to be processed on a single machine for delivery in batches to customers while, the jobs are ready at their release times, has been presented. This problem is a natural extension of minimizing the sum of flow times in presence of release dates by considering the possibility of delivering jobs in batches and introducing batch delivery costs. The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs. The extended problem arises in the context of coordination between machine scheduling and a distribution system in a supply chain network.

The branch and bound algorithm proved to be very efficient. Indeed, it proved to be far more efficient than the only existing algorithm for solving the problem, which is based on dynamic programming. This efficiency is attributable to the sharpness of the lower bounds derived, in addition to the high quality of an initial upper bound found using an effective heuristic.

Both lower bound and the upper bound are based on a careful analysis of the structural properties of the problem.

Chapter 5

5. Minimizing the sum of flow times (completion times) for the combined problem

In this chapter we consider the situation when a supplier and one of the manufacturers (assume manufacturer 1) cooperate together to minimize the sum of flow times plus delivery times over the system. Thus, the problem being considered is that of scheduling a set of jobs to be processed on a single machine by the supplier for delivery to several manufacturers in batches, and also scheduling the jobs which are to be processed on a single machine by manufacturer 1 for delivery of the final product to several customers in batches.

This problem can be considered as a combination of two problems. The first problem is a natural extension of a single machine scheduling problem where a set of jobs to be processed by supplier for delivery in batches to manufacturers 2 to n. The second problem is a natural extension of the two-machine flow shop problem, where there are some jobs to be processed by the supplier for delivery in batches to manufacturer 1 who has to process the jobs for delivery of the final products in batches to several customers. The combined problem arises in the context of

coordination between machine scheduling and a distribution system in a supply chain network.

Structural properties of the problem are investigated and used to devise a branch and bound solution scheme. Computational experience shows the efficiency of the algorithm and the benefit of cooperation in reducing total system cost over the system.

5.1 Introduction

This chapter considers a combined problem that occurs in a real world supply chain network, where a supplier makes deliveries to several manufacturers, M_g for $g = 1,...,G$ who in turn make deliveries to several customers. We address the condition that supplier and one manufacturer, M_1, cooperate together to minimize total flow times plus total delivery times over the system. Thus, there are single-stage jobs that the supplier processes and delivers them to some manufacturers $M_2,...,M_G$ in batches, and there are also two-stage jobs that the supplier processes and delivers them in batches to manufacturer M_1, who has to process the jobs and deliver the final products in batches to several customers. The objective is to minimize the total flow times (completion times) plus delivery times over the system. The two-stage problem is a natural extension of two-machine flow shop problem by considering the possibility of delivering jobs in batches and introducing batch delivery costs. The two-stage problem is the recognition of the $F2||\sum F_j$ or $F2||\sum C_j$ classical problem with additional term. The classical problem is NP-complete [43], and then it is easy to show that the recognition version of the extended problem is also NP-complete. The *permutation* version of the classical problem has been first studied by Ignall and Schrage [58], who presented a branch and bound approach based on two lower bounds.

Van de Velde [88] developed a branch and bound method based on applying the Lagrangian relaxation on the constraint that requires for each job the second operation starts after the completion of the first operation, and showed that his lower bound dominated both bounds suggested by Ignall and Schrage.

F. Della Cross et al. [34] considered several known lower bounds and introduced two new ones among them. They have tested these bounds on problem instances with up to 30 jobs. The computational results indicate that a new bound presented by them when applied jointly with the Van de Velde's lower bound [88], gave the best performing lower bounding procedure. F. Della Cross et al. [33] presented an enhancement of Van de Veld's lower bound by exploring sufficient conditions for the optimality of a given sequence when maximizing the Lagrangian dual problem. They report the computational results of a problem with up to 45 jobs.

Can Akkan and Selcuk Karabati [2] present a new lower bound calculation scheme based on a minimum-cost network flow formulation of the problem, which when integrated into a branch and bound algorithm that uses dominance criteria already established in the literature, can solve problems with as many as 60(45) jobs when processing times are uniformly distributed in the [1,10]([1,100]) range.

Two-machine flow shop problems that address an optimal value, with both machine scheduling and delivery cost, appear to be rather complex, though they are more practical than those which involve just one of those factors. This type of combined optimization is often encountered when a real-world supply chain management is under consideration. However, although there is a large body of research on the classical version of the problem, only a few articles address combined optimization problem that seek to coordinate machine scheduling with delivering jobs in batches. The complexity of some combined problems such as makespan and completion times, when the jobs are to be delivered after their processing time to customer or warehouse are measured by Lee and Chen [64]. Hurink and Knust [57] considered a flow shop problem for minimizing makespan with transportation times, but they have assumed that a single robot, which can shift only one job at a time, does all the transportations.

Hall and Potts [49] consider a variety of scheduling, batching and delivery problems that arise in an arborescent supply chain, where a supplier makes deliveries to several manufacturers, who also make deliveries to customers. One of the problems identified by Hall and Potts is when the supplier and one manufacturer cooperate together in order to minimize sum of flow times plus delivery costs for the entire system including the supplier and manufacturer. Hall and Potts derive a forward dynamic programming algorithm for the problem under the assumptions of *total SPT within groups*. By this assumption they assume that two stage-jobs for each

customer are sequenced by both i.e. supplier and manufacturer, in SPT order according to the total processing time of jobs on the supplier's machine and manufacturer's machine. They also make the further assumption of *total SPT within groups*, in which jobs for each of the manufacturers $M_2,...,M_G$, are sequenced in SPT order according to the processing time of jobs on the supplier's machine. Hall and Potts have proved that the overall time complexity of their algorithm for finding an optimal schedule is $O(n^{2G+7H-2})$ time while n, G and H identify the number of jobs, manufacturers and customers respectively.

In this chapter we consider the similar problem under assumptions that, the single –stage jobs for each of the manufacturers $M_2,...,M_G$ are sequenced in SPT order by the supplier according to the processing time of jobs on the supplier's machine, and that the two-stage jobs for each customer are also sequenced in SPT order by the supplier according to the processing time of jobs on the supplier's machine. We also assume that each batch for manufacturer M_1 contains the jobs that are destined for a particular customer, i.e. the jobs that are destined for different customers neither make a batch on the manufacturer's machine nor on the supplier's machine. This assumption is a natural assumption within the framework of supply chain management. This condition may be enforced as a part of the coordination between the supplier (upstream stage) and the manufacturer.

In what follows, we consider this problem, study its structural properties, derive upper and lower bounds, offer a branch and bound scheme for solving it and present the efficiency of algorithm for solving the problem instances up to 30 jobs. Furthermore the benefit of cooperation between the supplier and manufacturer will be presented.

5.2 Problem Definition

Let $N^S = \{1,...,N\}$ denote the set of jobs to be scheduled by the supplier and to be delivered in batch to several manufacturers M_g for $g = 1$ *to* G. The supplier cooperates with one of the manufacturers (without loss of generality assume manufacturer 1) who has also to process the jobs and delivers them in batches to several customers C_h for $h = 1$ *to* H. We denote the supplier's machine and manufacturer's machine by M_S, and M_M respectively.

Let $N^1 = \{1,...,n^1\}$ denote the single-stage jobs that supplier processes and delivers to manufacturers $M_2,...,M_G$ in batches and $N^2 = \{1,...,n^2\}$ denote two-stage jobs that supplier processes and delivers in batches to manufacturer 1. The jobs in set N^1 need only one operation i.e. machine M_S. Each job in set N^2 consists of two operations where the first operation must be processed on machine M_S and the second operation on machine M_M with the same order (*Permutation schedule*); also the second operation cannot begin before the first operation is complete. Each job is available at time zero. Both machines can process at most one job at time. A group of jobs forms a batch for supplier (manufacturer), if all of these jobs are delivered to a single manufacturer (customer) at the same time. We also assume that each batch for manufacturer 1 contains the jobs, which are destined for a particular customer, i.e. the jobs that are destined for different customers make a batch neither on the manufacturer's machine nor, on the supplier's machine. The single-stage jobs for each of the manufacturers $M_2,...,M_G$ are sequenced in SPT order by the supplier according to the processing time of jobs on the supplier's machine, and the two-stage jobs for each customer are also sequenced in SPT order by the supplier and manufacturer 1 according to the processing time of jobs on the supplier's machine. Let J_n^S for $n = 1,...,n_1$ show the set of batches that are to be delivered to manufacturer M_1 and, J_n^M for $n = 1,...,n_2$ show the set of batches that are to be delivered to customers C_h for h = *1 to H.* When there is no ambiguity we simplify J_n^S and J_n^M to j^S and j^M, respectively. The objective function that we consider is to minimize the sum of flow times plus delivery costs over the system. This is a combination of two problems. The first problem is a natural extension of $1||\sum F_j$ problem with additional term which is delivery cost for manufacturers $M_2,...,M_g$, i.e. the problem we considered in chapter 3, and the second problem is a natural extension of two-machine flow shop problem $F2||\sum F_j$ with additional term which is delivery cost for transporting jobs from the supplier to manufacturer M_1 and also delivery costs for each customer. Thus, using the standard classification scheme for scheduling problems [46], the objective

function is $1\|\sum F_j^C + \sum D_g^S y_g^S + \sum D_h^M y_h^M$, where y_g and y_h denote the number of deliveries for each manufacturer and customer respectively and $F_j^C = F_j^S + F_j^M$, while F_j^S and F_j^M show the sum of flow times on machines M_S, and M_M respectively. As cited in the last section the classical version of two- machine flow shop problem, $F2\|\sum F_j$, is NP-complete, then the combined problem that contains this problem with some additional terms is also NP-complete.

We also use the following notations:

p_i^S, the processing time of job i on machine M_S,

p_i^M, the processing time of job i on machine M_M,

D_g^S, the delivery cost for delivery of batches from supplier to manufacturer g,

D_h^M, the delivery cost for delivery of batches from manufacturer 1 to customer h,

δ_j, the size of batch $j \in J^S$ or $j \in J^M$,

A_j, the sum of processing time of jobs within batch $j \in J^S$ on machine M_S,

C_{1j}, the completion time of batch $j \in J^S$ on machine M_S which is equal to the completion time of each job within respective batch,

X_{1j}, the sum of completion times of jobs within batch $j \in J^S$ on machine M_S which is equal to $\delta_j C_{1j}$,

Ω_{1j}, the sum of flow times of jobs within batch $j \in J^S$ on machine M_S.

Although the size and substances of batches on set J_n^S and J_n^M are not the same, however it is efficient to mark the group of jobs that are delivered in a batch to machine M_M with the same index that it has on machine M_S. Therefore, we define variable B_j such that it shows the sum of processing time of jobs on machine M_M, which are delivered in batch $j \in J^S$ to manufacturer 1. C_{2j}, Ω_{2j}, and X_{2j} are defined analogously for machine M_M. We note that $X_{1j} = \Omega_{1j}$, while X_{2j} and Ω_{2j} are not identical. More clearly, assume p_1^S and p_2^S are processing times of first 2

jobs that have been scheduled on machine M_S and delivered to manufacturer 1 in one batch. The processing time of these jobs on machine M_M are p_1^M and p_2^M respectively and they have been processed and delivered to respective customer by two separate batches, then we have:

$X_{1j} = \Omega_{1j} = 2(p_1^S + p_2^S)$,

$\Omega_{2j} = 2p_1^M + p_2^M$,

$X_{2j} = 2(p_1^S + p_2^S) + 2p_1^M + p_2^M$, or $X_{2j} = X_{1j} + \Omega_{2j}$.

5.3 Structural Properties

In this section, structural properties of the problem, used subsequently to derive upper and lower bounds, are analyzed. We first rewrite the first proposition of the third chapter in term of completion time, then we provide and prove a set of propositions and corollaries that are useable in a two-machine flow shop problem and finally we extend them for a combined problem. Since any result that is proved for completion time applies also to flow time, we state and prove the following propositions and properties in terms of completion time.

Proposition 5.3.1. *For a set of batches to be scheduled on a single machine (machine M_S), the sequence ordered by the Shortest Effective Batch Time (SEBT) is optimal in terms of total completion time, with batch effective time $T_b = \frac{A_b}{\delta_b}$, where A_b is the total processing time of the batch and δ_b is the batch size (number of jobs in the batch), which could be equal to 1.*

Proof: See proposition 3.1 in chapter 3. □

In accordance with our description of the problem, each machine can process the jobs while the machine is free. In the absence of batch delivery, relaxing this constraint for the second machine such that the machine to be always free for newly arrived jobs and sorting the jobs in SPT on the first machine yields a lower bound in term of sum of flow times (completion times) for two-machine flowshop problem, see Ignall and Schrage [58] and also Hoogeveen and Kawaguchi [55]. In what

follows we will improve this idea to derive a lower bound for two-machine flowshop problem in presence of batch delivery.

Proposition 5.3.2. *For scheduling a set of batches in a two-machine flowshop, while the second machine is always free for processing the jobs of the newly arrived batch, the sequence ordered by the Shortest Effective Batch Time (SEBT) on the first machine, i.e. machine M_S, yields a lower bound in terms of total completion time such that if σ^* be an optimal schedule, then $\sum_{j=1}^{n} X_j(\sigma^*) \geq \sum_{j=1}^{n} X_{1j} + \sum_{j=1}^{n} \Omega_{2j}$, where $\sum_{j=1}^{n} X_j(\sigma^*)$ shows the total completion time.*

Proof.

$C_{11} = A_1$, $C_{21} = A_1 + B_1$,

$X_{11} = \delta_1 A_1$, $X_{21} = X_{11} + \Omega_{21}$ and

$C_{1j} = C_{1,j-1} + A_j$, $C_{2j} = \max\{C_{1j}, C_{2,j-1}\} + B_j$,

$X_{1j} = \delta_j C_{1j}$, $X_{2j} = \delta_j \max\{C_{1j}, C_{2,j-1}\} + \Omega_{2j}$

Consider any schedule σ. The lower bound will be found as follows

$X_{2j} = \delta_j \max\{C_{1j}, C_{2,j-1}\} + \Omega_{2j} \geq \delta_j C_{1,j} + \Omega_{2,j}$ which implies that

$X_{2j} \geq X_{1,j} + \Omega_{2,j}$ for $j = 1,...,n$; hence,

$$\sum_{j=1}^{n} X_j = \sum_{j=1}^{n} X_{2,j} \geq \sum_{j=1}^{n} X_{1,j} + \sum_{j=1}^{n} \Omega_{2,j}.$$

The term $\sum_{j=1}^{n} \Omega_{2,j}$ in the right side of the above equation is independent of the order of batches on machine M_S. More clearly, if the order of jobs within batches on the first machine does not change, the order by which the batches will be delivered to manufacturer 1 does not affect on term $\sum_{j=1}^{n} \Omega_{2,j}$.

The term $\sum_{j=1}^{n} X_{1,j}$ in the above equation yields:

$$\sum_{j=1}^{n} X_{1,j} = X_{1,1} + X_{1,2} + ... + X_{1,n} \text{ or}$$

$$\sum_{j=1}^{n} X_{1,j} = \delta_1 A_1 + \delta_2 (A_1 + A_2) + ... + \delta_n (A_1 + ... A_n)$$

that states the total completion times of jobs on the first machine, i.e. machine M_S. This term is minimized by virtue of proposition 5.3.1 and it completes the proof. □

The following corollary then follows immediately.

Corollary 5.3.1. *For a set of batches to be scheduled in two-machine flowshop, the sequence ordered by the Shortest Effective Batch Time (SEBT) on the first machine is optimal in terms of total completion time, if* $A_{i+1} > B_i$ *for all batches in the sequence.* □

Proposition 5.3.3. *In a partial schedule, where some batches have been formed on system, i.e. the jobs that are scheduled and decision about batching has been taken on both machines, but no decision has been taken yet on batching the remaining 'un-batched' jobs, a lower bound on the sum of completion times of an optimally completed schedule corresponds to the sum of completion times in a schedule formed by considering each un-batched job as a single-job batch for each machine and sequencing all batches in the order of SEBT on machine* M_S.

Proof: Any batching of un-batched jobs to complete the schedule will necessarily delay some jobs. Hence, considering each such job as a single-job batch ensures no delay. Thereafter, SEBT sequencing on machine M_S ensures that the resulting schedule minimizes total completion time by virtue of proposition 5.3.2. □

Proposition 5.3.4. *In an optimal solution, any batch that is scheduled on first machine that has* $\delta_b > 1$ *jobs will have the property that* $(\delta_b - 1)p_l^S < D_1^S$, *where* δ_b *shows the batch size, l is the last job in the batch and* D_1^S *is batch delivery cost to manufacturer 1.*

Proof: (by contradiction) Consider a batch that does not have the indicated property. Removing the last job and delivering it in a batch of its own will decrease the overall objective function by $(\delta_b - 1)p_l^S - D_1^S$, no matter where the batch happens to be in the sequence of batches. It is also worth noting that since the jobs in a batch are all delivered at the batch delivery time; the order of the jobs within a batch is immaterial. However, from the viewpoint of supplier (first machine), it is assumed

that these jobs are ordered according to SPT, then it is clear that the last job in each batch has the greatest processing time. □

Proposition 5.3.5. *In an optimal solution, any batch that is scheduled on the second machine that has* $\delta_b > 1$ *jobs will have the property that* $(\delta_b - 1)p_l^M < D_j^M$ *, where* δ_b *shows the size of batch,* l *is the last job in the batch and* D_j^M *is batch delivery cost to the corresponding customer.*

Proof: The proof is the same as cited for proposition 5.3.4. The only difference is that from the viewpoint of manufacturer (second machine) the jobs within a batch are not necessarily ordered according to SPT rule.□

The following corollaries then follow immediately.

Corollary 5.3.2. *In an optimal solution, any job on the first machine (machine* M_S*) that has a processing time greater than the batch delivery cost to the second machine (machine* M_M*) i.e., such that* $p_i^S > D_1^S$*, will form a single-job batch.* □

Corollary 5.3.3. *In an optimal solution, any job on the second machine (machine* M_M*) that has a processing time greater than the batch delivery cost to the corresponding customer i.e., such that* $p_i^M > D_h^M$*, will start a new batch.* □

We note that the difference between corollaries 5.3.2 and 5.3.3 stems from this fact that the jobs on machine M_M are not ordered according to SPT, then it may occasionally happen that a job that has a processing time greater than the batch delivery cost to the corresponding customer starts a new batch and the other jobs join to this batch for establishing a batch with greater size. However, if a batch is already started, a job that has a processing time greater than the batch delivery cost to the corresponding customer cannot join to this batch.

Proposition 5.3.6. *A lower bound on the number of batches that are scheduled on the first machine and destined for the second machine in an optimal solution can be found by the following greedy maximum batching algorithm; take the jobs destined*

for the customer concerned in SPT order; if the job may be added to the current batch by virtue of proposition 5.3.4, then add it; else start a new batch.

Proof: In accordance with our assumption, jobs have to be taken in SPT order on the supplier's machine. Since each batch is augmented until it can take no more jobs, the number of batches is minimized. Moreover, since this is done while relaxing (forgetting) the constraints of interaction with jobs for other manufacturers due to batching, the number found is a lower bound, as claimed. □

Proposition 5.3.7. *A lower bound on the number of batches that are scheduled on the second machine and destined for a customer in an optimal solution can be found by the following greedy maximum batching algorithm; take the jobs destined for the customer concerned in the same order that these jobs are delivered by supplier (permutation schedule); if the job may be added to the current batch by virtue of proposition 5.3.5, then add it; else start a new batch.*

Proof: In accordance with our assumption, jobs on the manufacturer's machine have to be taken in the same order with the supplier's machine (permutation schedule). Since each batch is augmented until it can take no more jobs, the number of batches is minimized. Moreover, since this is done while relaxing (forgetting) the constraints of interaction with jobs for other customers due to batching, the number found is a lower bound, as claimed. □

In carrying out the search, we will be continually evaluating the worth of moves that add a job to a preceding batch for the same customer on the supplier's machine. It is, therefore, important that this evaluation be carried out in a computationally efficient way. The following proposition helps achieve that aim.

Proposition 5.3.8. *Let a job k be in position s_r to the right of a batch b, which is in position s_l in a SEBT sequence on machine M_S (in which job k constitutes a single-job batch). If k may be added to b by virtue of proposition 5.3.4, then the change in the sum of job completion times resulting from doing so, could be found by updating the contribution of the batches between s_l and s_r inclusive.*

Proof: See the proof of proposition 3.7 in chapter 3. □

In consequence of the above proposition, evaluating the worth of a job joining an earlier batch can be calculated efficiently, provided the delivery time, the number of jobs and the contribution of each batch to total flow time are kept updated.

In completing a partial schedule, where some batches on machine M_S or M_M have been formed, but no decision has been taken yet on batching the remaining 'un-batched' jobs, a 'batching penalty', Δ_k, attaches to each un-batched job, since if the job is added to the last formed batch for the customer concerned total completion time (flow time) will increase, and if a new batch is started with it an additional batch delivery cost will be incurred. The following propositions help to achieve that aim for each machine in turn.

Proposition 5.3.9. *Batching penalty, Δ_k, attaches to each un-scheduled job on machine M_S such that $\delta_l p_k^S \le \Delta_k \le D_1^S$ where, δ_l is the number of jobs in the batch that it may join and D_1^S is the batch delivery cost to second machine (machine M_M).*

Proof: See proof of proposition 3.8 in chapter 3.□

Proposition 5.3.10 *Let unscheduled jobs on machine M_M and the last batch for the same customer both belong to a group of jobs that are delivered by supplier in the same batch. Then, batching penalty, $\Delta_k = \delta_l p_k^M \le D_h^M$, attaches to each unscheduled job on machine M_M, where δ_l is the number of jobs in the batch that it may join and D_h^M is the appropriate batch delivery cost.*

Proof: Consider a partial schedule S. Since job k and the last batch l for the same customer both belong to a group of jobs that are delivered by supplier in the same batch, and since according to our assumption each batch only contains the jobs that are destined for a particular customer, and because it is a permutation schedule problem, then job k must be next to the last job of batch l, i.e. no job exist between job k and the last job of batch l. If k may not be added to l by virtue of proposition 5.3.5 or is not added to it, then a penalty equal to D_h^M will be incurred. Otherwise, total flow time will increase. To achieve this aim it is sufficient to compare the total

flow time of partial schedule before and after of joining k and l. We note that the flow times of all the batches preceding l and k will remain unaffected and so will the flow times of all the succeeding batches. Let T and δ_l show the flow time and size of batch l on machine M_M respectively.

The total flow time of partial schedule on machine M_M before job k is added to batch l is:

$$\delta_l T + (T + p_k^M), \qquad (1)$$

The total flow time of partial schedule on machine M_M after job k is added to batch l is:

$$(\delta_l + 1)(T + p_k^M) \qquad (2)$$

Comparing terms of equations (1) and (2), we achieve the batching penalty;

$\Delta_k = \delta_l p_k^M$ which completes the proof. □

Proposition 5.3.11 *Let unscheduled jobs on machine M_M and the last batch for the same customer belong to two different groups of jobs which are delivered to machine M_M separately. Batching penalty; $\delta_l p_k^M \le \Delta_k \le D_h^M$ attaches to each un-scheduled job where, δ_l is the number of jobs in the batch that it may join and D_h^M is the appropriate batch delivery cost.*

Proof: Consider a partial schedule S which is sequenced according to SEBT on machine M_S such that $\frac{A_a}{\delta_a} \le \frac{A_{a+1}}{\delta_{a+1}} \le \ldots \le \frac{A_{b-1}}{\delta_{b-1}} \le \frac{A_b}{\delta_b}$. Let job k be the first unscheduled job on the manufacturer's machine, which is delivered to machine M_M by single-job batch j_b^S . This is while the last batch with the same customer i.e. batch l, belongs to a group of jobs that has been delivered to machine M_M previously by batch j_a^S . If k may not be added to l by virtue of proposition 5.3.5 or is not added to it, then a penalty equal to D_h^M will be incurred. Otherwise, total flow time will increase. To access this increase, without loss of generality let each group of jobs that are delivered to machine M_M through batches j_a^S to j_b^S make a batch on machine M_M with the same size to the corresponding batch such that $\delta_a = \delta_l, \delta_{a+1} = \delta_{l+1}, \ldots$ *and*

$\delta_{b-1} = \delta_{l+b-a-1}$ while, $\delta_a,...,\delta_b$ are the sizes of batches j_a^S to j_b^S on machine M_S and $\delta_l, \delta_{l+1},..., \delta_{l+b-a-1}$ are the sizes of batches j_l^M to $j_{l+b-a-1}^M$ on machine M_M.

Four cases need to be distinguished:

Case 1 Batch j_b^S follows batch j_a^S directly ($b=a+1$):

$$\Delta_k = (A_a + \max\{A_b, B_a\} + p_k^M)(\delta_a + 1) - (A_a + B_a)\delta_a - (A_a + A_b + p_k^M)$$

$$\geq \delta_a p_k^M$$

Case 2 $b > a + 1$ and after the newly formed batch is established, batch j_b^S moves to the position next to batch j_a^S:

$$\Delta_k = (A_a + \max\{A_b, B_a\} + p_k^M)(\delta_a + 1) + (A_a + A_b + A_{a+1} + B_{a+1})\delta_{a+1} + ...$$

$$-(A_a + B_a)\delta_a - (A_a + A_{a+1} + B_{a+1})\delta_{a+1} - ...(A_a + A_{a+1} + ...A_b + p_k^M) \geq \delta_a p_k^M$$

Case 3 $b > a + 1$ and after the newly formed batch is established batch j_a^S moves to the end of the subsequence, i.e., after the batch that was in position j_{b-1}^S:

$$\Delta_k = (A_{a+1} + B_{a+1})\delta_{a+1} + ... + (A_{a+1} + ... + A_a + \max\{A_b, B_a\} + p_k^M)(\delta_a + 1) -$$

$$(A_a + B_a)\delta_a - (A_a + A_{a+1} + B_{a+1})\delta_{a+1} - ...(A_a + A_{a+1} + ...A_b + p_k^M) \geq \delta_a p_k^M$$

Case 4 $b > a + 1$ and after the newly formed batch is established batch j_a^S and batch j_b^S move to a position in between, say, after the batch that used to be in position a + α:

$$\Delta_k = (A_{a+1} + B_{a+1})\delta_{a+1} + ... + (A_{a+1} + ... + A_{a+\alpha-1})\delta_{a+\alpha-1} +$$

$$(A_{a+1} + ... + A_{a+\alpha-1} + A_a + \max\{A_b, B_a\} + p_k^M)(\delta_a + 1) +$$

$$(A_{a+1} + ... + A_{a+\alpha-1} + A_a + A_b + A_{a+\alpha+1} + B_{a+\alpha+1})\delta_{a+\alpha+1} + ... +$$
$$(A_{a+1} + ... + A_{a+\alpha-1} + A_a + A_b + A_{a+\alpha+1} + ... + A_{b-1})\delta_{b-1} -$$

$$(A_a + B_a)\delta_a - ... - (A_a + A_{a+1} + ... + A_\alpha + B_\alpha)\delta_\alpha - ... -$$

$$(A_a + A_{a+1} + ...A_b + p_k^M) \geq \delta_a p_k^M$$

Since $\delta_a = \delta_l$, then $\Delta_k \geq \delta_l p_k^M$ which completes the proof. □

Proposition 5.3.12. *All propositions and corollaries cited above for two-machine flowshop problem are extendable to the combined problem.*

Proof. For the combined problem, there are some jobs that require machine M_S only, i.e., the jobs that are to be processed and delivered in batch to manufacturers $g = 2,...n$. In accordance to our description of the problem, each batch, which is constructed by the supplier for delivery to manufacturer 1, contains only the jobs that are destined for a particular customer. Hence, it can be assumed that all jobs, including the jobs that are to be processed for manufacturers $g = 2,...n$, need two operations while the processing times of some jobs on the second machine (machine M_M) are zero. By this interpretation of the problem, the supplier has to process and deliver all jobs in batch to manufacturer 1 while, the batch delivery cost for transporting batches from supplier to manufacturer 1 (second machine) is dependent on the final destination of jobs, i.e. customers. On the other hand, manufacturer 1 has to process and deliver jobs in batches to several customers while, the processing time of some jobs on the manufacturer's machine (second machine) are zero, and also the delivery times of these jobs, i.e. the jobs with processing time of zero on the second machine, are zero too.

Carrying out in this manner the combined problem reduces to a two-machine flowshp problem and as a result all propositions and corollaries proved above are also valid for the combined problem. □

Henceforward we will distinguish between the jobs according to their destination as follows. The jobs are destined for a *regular customer* when both operations of jobs on the first and second machines are not zero. In contrast, the jobs are destined for a *virtual customer* when their second operations are zero. Whenever we use *customer,* without any prefix, we mean both *regular* and *virtual customer.*

As a consequence of the above description and proposition, the combined problem will reduce to a two-machine flowshop problem in the following way.

There is the set of jobs $N^S = \{1,...,N\}$ that are to be scheduled for several consumers C_λ on both supplier's machine and manufacturer's machine for $\lambda = G + H - 1$, where G is the number of *virtual customers* (in fact

manufacturers $g = 2,...n$) and H are the number of *regular customers*. D_λ^S and D_λ^M define the delivery cost to manufacturer 1 and to customers respectively while, the delivery costs for transporting batches from supplier to manufacturer 1 varies for different customers and, some of the delivery costs from manufacturer 1 to customers are zero.

5.4 Branch and Bound Scheme

Search of the solution space is structured as a triplet 0-1-2 search tree, where each node is partitioned into three: one indicating that a job on machine M_S is added to the last batch for the customer concerned; also if this job belongs to a regular customer, then the decision about adding it to the last batch for the regular customer concerned will be taken by virtue of proposition 5.3.5 and corollary 5.3.3 (1); the second indicating the start of a new batch on machine M_S and if this job belongs to a regular customer, then it is added to the last batch for the regular customer concerned, i.e. the condition cited in propositions 5.3.5 is certainly satisfied (2); and the third indicating the start of new batch on both machines M_S and M_M that could be a single-job batch on both machines (0).

We note that in case (1), when a job belongs to a regular customer, i.e. its processing time on the second machine is not zero, two options may arise. The first option is when the job is added to the last batch on the first machine and it is also added to the last batch on the second machine, and the second one is when the job is added to the last batch on the first machine but it is not added to the last batch on the second machine. Since considering both options in one branch can reduce the tasks of search tree we have considered both options in one branch.

It is also worth noting that in case (2), although the job under consideration does not contribute to making a batch on machine M_S, it may still be moved from its original position. Furthermore, since the job under consideration joins the last batch with the same regular customer to make a grater batch, then the flow time of such job must be calculated as a member of the newly created batch. This implies that although this job indicating the start of a new batch on machine M_S, it still contributes as an element of a batch on the system in respect of total flow time (completion time) consideration.

The tree is constructed in a depth-first fashion. Other components of the branch and bound scheme are presented in the following subsections.

5.4.1 Branching and ordering of variables

Variables are ordered in accordance with the SPT of the corresponding jobs on machine M_S. At each node of the decision tree, two tasks are performed. First, variables that have to be set to zero, because no batch for the customer concerned has been formed or by virtue of propositions 5.3.4 and 5.3.5, are set to zero. Secondly, the first free variable (free variables are the ones that have not yet been committed to either zero or one or two) in the SPT sequence is set to one.

5.4.2 Fathoming and backtracking

A node is fathomed if:

Either it is a leaf node, i.e., all variables are fixed.

Or the lower bound exceeds or equals the incumbent upper bound.

Fathoming initiates backtracking to the first node associated with a variable whose value is either 1 or 2. If the value of this variable is 1 then it is set to 2; else it is set to zero. If no such node is found, the search terminates.

5.4.3 Upper bounds

Unfortunately, due to the complexity of the problem neither the algorithms, which were used in the third chapter, i.e. maximization heuristic and multi-start greedy heuristic, nor the second algorithm that was used in the forth chapter, can be generalized for this problem. However, the first algorithm of the last chapter can still be modified for application to the problem in hand.

It is worth noting that since both the single–stage jobs and two-stage jobs for each customer are sequenced in SPT order according to processing times of jobs on the supplier's machine then, applying any algorithm for sequencing the jobs on the second machine, before scheduling of jobs on the first machine, cannot lead to a efficient result. Consequently, the algorithm that is provided here is based on sequencing the jobs on the first machine. Whenever a batch on the first machine is constructed or a new single-job batch is added to the last batch, then the possibility of establishing a batch on the second machine will also be considered.

Algorithm *UB*—initial upper bound
Begin

• Form all jobs into single-job batches and sequence the batches in SEBT order (which, in this case, is equivalent to SPT) on the supplier's machine, and call it original sequence.

For i =1 to number of batches **do**

• Select the first single-job batch in the sequence, which is not selected yet.

For $j = i + 1$ to number of batches **do**

• Scan forward until the first single-job batch with the same customer to the selected batch, i.e. batch i, is found and call it k.

• Move the newly found single-job batch, i.e. batch k, to the position of batch i, and join them to make a bigger batch, if they will join profitably, otherwise, if they belong to a regular customer, move single-job batch k to the position next to i. In either case, regardless of whether they are joined or not, consider the possibility of joining the corresponding single-job batches on the second machine (if they belong to a regular customer) by virtue of proposition 5.3.5 and corollary 5.3.3 and join them, if they will join profitably. Return job k to its original position if no advantage is yielded.

• Continue the interior loop until no improving move is found.

• Continue the exterior loop until a complete scan of all batches.

end.

5.4.4 Lower bounds

It is worth recalling that at each node of the decision tree, if, in view of the batching decisions already taken, a job has to start a new batch on machines M_S or M_M, then the partial solution is immediately augmented by a batch starting with that job.

At each node of the decision tree, a lower bound on total flow time is calculated in accordance with proposition 5.3.3. Additionally, batch delivery costs are added for each batch already formed.

Furthermore, a lower bound on the batching penalties is calculated by applying the logic of propositions 5.3.9, 5.3.10 and 5.3.11 in the following way. The first un-batched job, k, on the supplier's machine (manufacturer's machine) will either start a new batch or will join the last batch. It would, therefore, attract a lower bound on its batching penalty = $\delta_l p_k^S$ ($\delta_l p_k^M$), where δ_l is the number of jobs in the last batch on the supplier's machine (manufacturer's machine). Since for each subsequent job, we do not know the number of jobs in the batch that it may join in the optimal completion of the current partial solution, each such job would attract a lower bound on its batching penalty = p_k^S (p_k^M) (i.e., the lowest batching penalty that it may incur be if it joined a single-job batch).

The overall lower bound is then the sum of the lower bound on the total flow time, batch delivery costs of the batches already formed and the sum of the lower bounds on the batching penalties for the un-batched jobs of each machine.

5.4.5 Optimum value at Leaf nodes

Proceeding with carrying out the branch and bound scheme explained above leads us to the all feasible combinations of batching at leaf nodes. Since these combinations are made by virtue of proposition 5.3.2, which is based on relaxing over one of the constraints on the second machine, i.e. the constraint that does not allow processing more than one job at the same time on the manufacturer's machine, then those combinations must be considered again without applying the relaxation. However, at leaf nodes we have a set of batches for which decisions about their batching have been taken but the optimality of the sequence is not guaranteed. Each sequence at leaf node can be considered again in the following way.

Let ϕ reperesents the total flow time plus delivery time of a sequence at leaf node, without applying any relaxation.

- If ϕ is less than current upper bound and the sequence is ordered by logic of corollary 5.3.1, then this value is certainly a local optimum. For this case the former

upper bound replaces the later (if the later is sharper) and no more consideration is need.

- If ϕ is less than current upper bound but the sequence is not ordered by logic of corollary 5.3.1, this value is also a local optimum but there is a need for more searching if there is any sharper value. For this case, the former upper bound replaces the sharper value (if it is sharper than current upper bound) at the end of consideration.
- Otherwise, there is a need for more seeking.

However, in the strong sense, our problem at the leaf nodes is reduced to a two-machine flowshop problem without batch delivery and with a property (corollary 5.3.1) that makes the problem easier. To achieve the optimum value, we can apply a branch and bound technique to the problem in the following way. Each node represents a sequence of r of the n batches, with $n-r$ of the batches remaining to be ordered. Each node does not have more than λ children while $\lambda = H + G - 1$ (see proposition 5.3.12). ϕ is our upper bound and the required lower bound on the sum of flow times for schedules emanating from a given node is roughly the following:

$$\Psi_r + \Psi'_{n-r},$$

where Ψ_r is the sum of the flow times of the r batches assigned in the partial schedule represented by node, Ψ'_{n-r} is the sum of flow times for the remaining $n-r$ batches, calculated under assumption that the remaining batches are sequenced in order of SEBT on machine M_S and $A_{i+1} > B_i$ for all i of the $n-r$.

5.4.6 Numerical example

Consider the following combined problem. There is a set of jobs to be scheduled by the supplier for delivery in batch to two manufacturers, say manufacturer 1 and 2, with delivery costs of 10 and 15 respectively. Furthermore, manufacturer 1 has to process and deliver jobs to two customers, customers1 and 2, with delivery costs of 20 and 30 respectively.

	Manufacturer 1				Manufacturer 2	
	Job 1	Job 2	Job 3	Job 4	Job 1	Job 2
Processing time on machine M_S	3	8	12	14	4	9

$D_1^S = 10$, and $D_2^S = 15$.

	Customer 1		Customer 2	
	Job 1	Job 2	Job 1	Job 2
Processing time on machine M_S	3	12	8	14
Processing time on machine M_M	7	10	6	11

$D_1^M = 20$, and $D_2^M = 30$.

By virtue of proposition 5.3.12, we can assume that manufacturer 2 is a virtual customer; the problem will be changed to the following problem:

	Customer 1		Customer 2		Customer 3(M2)	
	Job 1	Job 2	Job 1	Job 2	Job 1	Job 2
Processing time on machine M_S	3	12	8	14	4	9
Processing time on machine M_M	7	10	6	11	0	0

Initial lower bound:

The lower bound on total flow time, *LBF*, corresponding to flow time under SPT on the supplier's machine = 169. Batch delivery costs, *LBD*, is batch delivery cost on machine M_S, *LBD1*, plus batch delivery cost on machine M_M, *LBD2*. It is worth recalling that only the jobs that are destined for a particular customer can establish a batch. Therefore, with regard to machine M_S we have to start one batch for each regular or virtual customer which yields: *LBD1* = 10 + 10 + 15 = 35.

In line with our description of the problem, the delivery cost of transporting batches from machine M_M to the virtual customers is zero thus, $LBD2 = 20 + 30 + 0 = 50$, and $LBD = LBD1 + LBD2 = 85$.

The lower bound on batching penalties, *LBB* is the sum of *LBB1*, i.e. batching penalties on machine M_S, and *LBB2,* which is batching penalties on machine M_M. *LBB1* is: (10) + (10) + 9 = 29, where the numbers within the first (second) bracket correspond to the penalties incurred by the only remaining job of the first (second) regular customer in turn, and the third number corresponds to the penalty incurred by the only remaining job of the virtual customer. Note that the 10 for the first (second) regular customer results from substituting the initial lower bound on the batching penalty, which are equal to 12 and 14, by the batch delivery costs, by virtue of the fact that the minimum number of batches for the first (second) regular customer is two. *LBB2* is: 10 + 11 = 21, where the first and second numbers correspond to the penalties incurred by the remained jobs of regular customers in turn. Hence,

$LBB = LBB1 + LBB2 = 50$, and

$LB = LBF + LBD + LBB = 169 + 85 + 50 = 304$.

Initial upper bound:

To differentiate between batches on the first and second machines we enclose batches belonging to machine M_M by curl bracket, $\{\ \}$. Furthermore, the group of jobs enclosed by symbol [] indicate that these jobs have not made a batch but they have been scheduled consecutively on machine M_S while they have made a batch on machine M_M.

Applying algorithm *UB* to the 6 single-job batches on the supplier's machine leads to the following schedule:

|(11)|, |(13) (23) |, [(12)|(22)], |(21)| {|(11)|, | (12)(22)|, |(21)| }, with a total flow time of 200. Adding total batch delivery costs gives UB* = 200 + 125 = 325.

Branch and Bound:

- S_0 = |(11) |, |(13)|, | (12)| { |(11) |, | (12)|}; $LBF = 169$,
  $LBD1 = 10+10+15 = 35$, $LBD2 = 20+30 = 50$, $LBD = 85$,
  $LBB1 = 10+10+9 = 29$, $LBB2 = 10+11 = 21$, $LBB = 50$, $LB = 304$.

- S_1 = |(11) |, |(13) (23)|, | (12)| { |(11) |, |(13) (23)|, | (12)|}; *LBF* = 179, *LBD1* = 10+15 + 10 = 35, *LBD2* = 20+30 = 50, *LBD* = 85, *LBB1* = 10+10 = 20, *LBB2* = 10 +11 = 21, *LBB* = 41, *LB* = 305.

- S_2 = |(13) (23)|, [(11) (21)], | (12)| { |(11) (21) |, |(12)|}; LBF = 205, *LBD1* = 15+10+10+10 = 45, *LBD2* = 20+30 = 50, *LBD* = 95, *LBB1* = 0 + 0 + 10 = 10, *LBB2* = 0 +11 = 11, *LBB* = 21, *LB* = 321.

- S_3 = |(13) (23)|, [(11) (21)], [(12)(22)] { |(11) (21) |, |(12)(22)|}; LBF = 224,*LBD1* = 15+10+10+10 +10 = 55, *LBD2* = 20+30 = 50, *LBD* = 105, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 329. LB > UB backtrack.

- S_4 = |(13) (23)|, [(11) (21)], (12)|, |(22)| { |(11) (21) |, |(12)|, |(22)|}; LBF = 205, *LBD1* = 15+10+10+10 +10 = 55, *LBD2* = 20+30 + 30 = 80, *LBD* = 135, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 340. LB > UB backtrack.

- S_5 = |(11) |, |(13)(23)|, |(12)|, |(21)| { |(11) |, | (12)|, |(21)|}; *LBF* = 179, *LBD1* = 10+15+10 +10 = 45, *LBD2* = 20+ 20 + 30 = 70, *LBD* = 115, *LBB1* = 0+0+10 = 10, *LBB2* = 0 +11 = 11, *LBB* = 21, *LB* = 315.

- S_6 = |(11) |, |(13)(23)|, [(12)(22)], |(21)| {|(11) |, |(12)(22)|, |(21)|}; *LBF* = 200, *LBD1* = 10+15+10 +10 +10 = 55, *LBD2* = 20+ 20 + 30 = 70, *LBD* = 125, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 325. LB = UB backtrack.

- S_7 = |(11) |, |(13)(23)|, |(12)|, |(21)|, |(22)|,{|(11) |,|(12)|, |(21)|, |(22)|}; *LBF* = 179, *LBD1* = 10+15+10 +10 +10 = 55, *LBD2* = 20+ 20 + 30 +30 = 100, *LBD* = 155, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 334. LB > UB backtrack.

- S_8 = |(11) |,|(13)|, |(12)|, |(23)|, {|(11) |, |(12)|, |(22)|}; *LBF* = 169, *LBD1* = 10+15+10 +15 = 50, *LBD2* = 20+ 30 = 50, *LBD* = 100,

LBB1 = 10 + 10 = 20, *LBB2* = 10 + 11 = 21, *LBB* = 41, *LB* = 310.

- S_9 = |(13)|, [(11) (21)], |(12)|, |(23)|, {|(11)(21)|, |(12)||}; *LBF* = 192, *LBD1* = 15+10 +10 +10 +15 = 60, *LBD2* = 20+ 30 = 50, *LBD* = 110, *LBB1* = 10, *LBB2* = 11, *LBB* = 21, *LB* = 323.

- S_{10} = |(13)|, [(11) (21)], |(23)|, [(12)(22)], {|(11)(21)|, |(12)(22)|}; *LBF* = 212, *LBD1* = 15+10 +10 +15 +10 + 10 = 70, *LBD2* = 20+ 30 = 50, *LBD* = 120, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 332. LB > UB backtrack.

- S_{11} = |(13)|, [(11) (21)], |(12)|, |(23)|,|(22)| {|(11)(21)|, |(12)|, |(22)|}; *LBF* = 192, *LBD1* = 15+10 +10 +10 +15 +10 = 70, *LBD2* = 20+ 30 +30 = 80, *LBD* = 150, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 342. LB > UB backtrack.

- S_{12} = |(11) |, |(13)|, | (12)|, |(23)|, |(21)| { |(11) |, | (12)|, |(21)|}; *LBF* = 169, *LBD1* = 10+15 10 + 15 +10= 60, *LBD2* = 20+30 +20 = 70, *LBD* = 130, *LBB1* = 10, *LBB2* = 11, *LBB* = 21, *LB* = 320.

- S_{13} = |(11) |, |(13)|, |(23)|, [(12)(22)], |(21)| { |(11) |, | (12)(22)|, |(21)|}; *LBF* = 191, *LBD1* = 10+15 +15 +10 +10 +10= 70, *LBD2* = 20+30 +20 = 70, *LBD* = 140, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 331.

- S_{14} = |(11) |, |(13)|, |(12)|, |(23)|, |(21)|, |(22)| { |(11) |, |(12)|, |(21)|, |(22)|}; *LBF* = 169, *LBD1* = 10+15 +10 +15 +10 +10= 70, *LBD2* = 20+30 +20 + 30 = 100, *LBD* = 170, *LBB1* = 0, *LBB2* = 0, *LBB* = 0, *LB* = 339.

- Search Completed.

Hence, the optimum value is 325, and the sequence that leads us to the upper bound is the optimum schedule. The corresponding sequence is illustrated in figure 4.1.

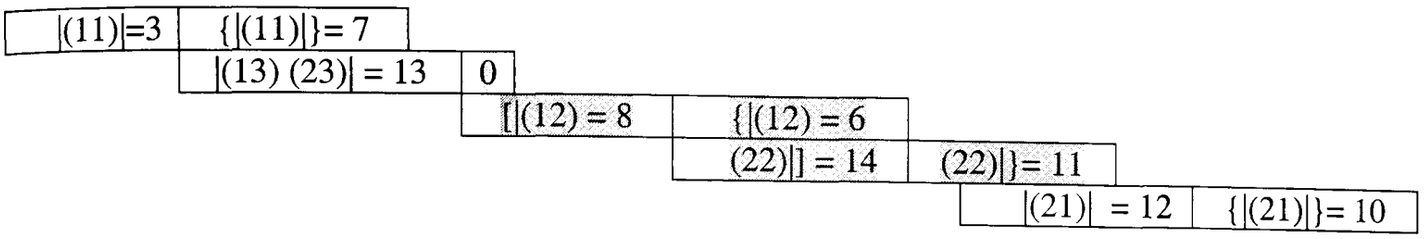


Figure 4.1: The procedure for achieving the optimum value.

The first cell and the second cell in each line show the processing time of batches on the first machine and the second machine respectively.

Symbols "[]" and "{}" are defined previously. The symbols are not closed until all jobs within a batch are processed.

5.4.7 Benefit of cooperation

The optimal schedule from the viewpoint of supplier in the absence of cooperation with manufacturer 1 can be obtained by applying the algorithm provided in chapter 3. The optimal sequence is |(11)|, |(21)(22)|, (21)|, |(31)|, |(41)| for total flow time plus delivery time of 200. In accordance to this sequence, the jobs will be delivered to manufacturer 1 by 4 single-job batches at times 3, 24, 36, and 50 respectively. These times identify the release dates of jobs for the manufacturer's problem. Hence, the manufacturer's problem can be shown as follows.

	Customer 1		Customer 2	
	Job 1	Job 2	Job 1	Job 2
Processing time on machine M_M	7	10	6	11
Release date	3	36	24	50

From the viewpoint of manufacturer 1 the optimal schedule is achieved by applying the algorithm provided in chapter 4. For manufacturer 1, the optimum value will be obtained by processing and delivering the jobs in 4 single-batches that are sequenced as |(11)|, |(12)|, |(21)|, |(22)| for total completion time plus delivery cost of 247. Subtracting the sum of release dates from this value will yield:

Total flow time plus delivery cost = 247 – 113 = 134.

Hence, the sum of total flow times plus delivery times over the system (STFD), i.e. for the supplier and manufacturer, before cooperation is:

STFD = 200 + 134 = 334.

Comparing this value and the optimum value i.e. 325, which is already obtained for the combined problem in the last section, shows that cooperation reduces the total system cost of this problem instance by 9, i.e. 2.69%.

It is worth noting that when the supplier acts independently job |(31)| proceeds job |(41)| while in the combined problem job |(41)| proceeds job |(31)|.

Further examples and considerations on the benefit of cooperation will be presented in the next chapter.

5.5 Computational Results

To consider the efficiency of the combined algorithm for solving the problems the algorithm was tested on a set of randomly generated problem instances. For the set, six subsets were generated, one with 2 manufacturers and 2 customers denoted by (2-2), second with 2 manufacturers and 3 customers (2-3), third with 3 manufacturers and 3 customers (3-3), fourth with 3 manufacturers and 4 customers (3-4), fifth with 4 manufacturers and 3 customers (4-3), and the sixth subset with 4 manufacturers and 4 customers (4-4). In each subset, the number of jobs was varied up to a total of 30. The jobs were randomly distributed among manufacturers and customers, with each being assigned at least two jobs. Processing times were randomly generated integers from the uniform distribution defined on [1,100]. For each distribution type and problem size 10 instances were generated. In order to limit the time taken by the procedures, a limited bound was placed on the number of nodes on the algorithm that finds the optimum at leaf nodes. The procedures were terminated when the number of nodes generated exceeded 80,000,000. In the tables 1-6, below, the minimum, maximum, average running times over the appropriate instances and the number of unsolved problems are represented. The average running times are calculated only over the solved problem instances.

The computational experiments were run on a Pentium 4 computer with 2.40 GHz of CPU and 512 MB of RAM. The B&B algorithm was coded in C++. The results are shown in tables 1 to 6.

As can be clearly seen, the Branch and Bound (B&B) algorithm derived for the combined problem can solve the problem instances with up to 22 jobs for all cases, and when the number of jobs is 27 only one instance in each of the subsets (2-3) and (2-4) is unsolved.

It is worth recalling that the only existing algorithm, with the same problem under consideration, is the dynamic programming algorithm developed by Hall and Potts. They have proved that the overall time complexity of their algorithm for finding an optimal schedule is $O(n^{2G+7H-2})$ time while n, G and H reperesent the number of jobs, manufacturers and customers respectively.

In contrast to the previous two chapters, here we have not coded the DP algorithm. This is mainly due to different set of assumptions made in the two approaches. In fact, the assumptions we have applied are a little more restricted than what they have applied, but on the other side the algorithm we have developed is more practical than theirs. Although the DP algorithm is not coded but we can still have an imagination about its running time for solving the problem instances. To achieve this gain we note that the time complexity of DP algorithm for the last problem, considered in chapter 4, was $O(n^{3H})$ time. We showed in the last chapter that the DP algorithm could solve the problem instances with less than 18 jobs while the number of customers was 4, or problem instances with less than 22 jobs while the number of customers was 3. Comparing the time complexity of two algorithms, i.e. the DP algorithms of the last problem and the current problem, makes it clear that the current DP algorithm can only solve the problem instances with only a few jobs when the number of manufacturers and customers are also very restricted. To make it more clear assume that the supplier has to supply the jobs only for one manufacturer, i.e. manufacturer 1. Then the time complexity of problem will reduce to $O(n^{7H})$ time. Let n_1 and n_2 show the number of jobs for the last problem and the current problem respectively. Roughly speaking, we can say that $n_2 = n_1^{3/7}$. Then, for the case that the number of customers is 3 the number of jobs for the current problem cannot exceed $n_2 = 22^{3/7}$ ($n_2 < 4$), while 22 is the maximum job numbers that DP algorithm could solve the problem instances with 3 customers for the last problem. Similarly for the case that the number of customers is 4, the number of jobs cannot

exceed $n_2 = 18^{3/7}$ ($n_2 < 4$). In other words, the DP algorithm cannot solve the problem instances for more than 1 manufacturer and 3 customers.

Now let's assume that the number of manufacturers is 2. Then, the time complexity of DP algorithm for the current problem will reduce to $O(n^{7H+2})$. For this case the rate of jobs in comparing with the last problem is $n_2 = n_1^{(3H/7H+2)}$. Thus, it can be clarified that when the number of manufacturers is 2 and the number of customers is 3, then the maximum number of jobs cannot exceed $n_2 = 22^{9/23}$ ($n_2 < 4$) which means again that DP algorithm cannot handle it.

In a similar way it can be estimated that the running times for the problem instances with 1 (2) manufacturers, 3 (2) customers and only 5 jobs, if assume that DP algorithm is used to solve it, may be huge that is so far even from the worst case running time which is reported in tables 1 to 6.

Moreover, for majority of instances the problem can be solved by B&B algorithm very fast. It is worth recalling that the combined problem is a multi variables problem, related to the distribution of jobs among manufacturers and customers, and also the processing times and delivery times of both customers and manufacturers. On the other hand the algorithm, which is provided for the combined problem, contains two sub-algorithms. The first algorithm considers all combinations of delivering jobs in batch and provides a set of acceptable combination of batches at leaf nodes while the second algorithm solves a two-machine flow shop problem for a set of batches. Due to the structure of the problem, it is understandable that our algorithm might readily solve some instances with total jobs in excess of 30, while for others the running times may be significantly longer. The great running time for some instances is due to the weakness of the second sub-algorithm, which applies on the set of batches at the leaf nodes.

Table 1: Running times for problem instances with 2 manufacturers and 2 customers.

Number of Manufacturers	Number of Customers	Number of Jobs	Running time (Per ms) Min	AVG	Max	Unsolved Instances
2	2	7	10	13	20	0
2	2	12	10	104	410	0
2	2	17	30	506	3024	0
2	2	22	50	11245	96719	0
2	2	27	40	219	1142	0
2	2	30	360	11191	36162	2

Table 2: Running times for problem instances with 2 manufacturers and 3 customers.

Number of Manufacturers	Number of Customers	Number of Jobs	Running time (Per ms) Min	AVG	Max	Unsolved Instances
2	3	12	10	40	120	0
2	3	17	20	5276	45355	0
2	3	22	40	817	5388	0
2	3	27	100	241133	937058	1
2	3	30	70	871871	404101	3

.

Table 3: Running times for problem instances with 3 manufacturers and 3 customers.

Number of Manufacturers	Number of Customers	Number of Jobs	Running time (Per ms) Min	AVG	Max	Unsolved Instances
3	3	12	10	33	70	0
3	3	17	20	2951	29122	0
3	3	22	30	10558	75248	0
3	3	27	30	516178	4306993	0
3	3	30	70	248566	1851943	1

Table 4: Running times for problem instances with 3 manufacturers and 4 customers.

Number of Manufacturers	Number of Customers	Number of Jobs	Running time (Per ms) Min	AVG	Max	Unsolved Instances
3	4	12	20	108	251	0
3	4	17	20	555	1692	0
3	4	22	80	429951	1997773	0
3	4	27	30	33674	151748	0
3	4	30	60	542592	4205367	1

Table 5: Running times for problem instances with 4 manufacturers and 3 customers.

Number of Manufacturers	Number of Customers	Number of Jobs	Running time (Per ms) Min	AVG	Max	Unsolved Instances
4	3	12	10	26	40	0
4	3	17	40	865	2153	0
4	3	22	20	403	3224	0
4	3	27	51	2136	6770	0
4	3	30	51	375060	2275161	2

Table 6: Running times for problem instances with 4 manufacturers and 4 customers.

Number of Manufacturers	Number of Customers	Number of Jobs	Running time (Per ms) Min	AVG	Max	Unsolved Instances
4	4	17	20	3847	33257	0
4	4	22	40	23044	130628	1
4	4	27	80	165235	1007989	1
4	4	30	120	119511	647050	4

5.6 Conclusion

A combined problem for scheduling a set of jobs to be processed on a single machine by supplier for delivery to some manufacturers in batches, and scheduling the jobs by one of the manufacturers on a single machine for delivery of the final products to some customers in batches has been presented. The combined problem has been considered as a natural extension of two problems, i.e. the problem of minimizing the sum of flow times on a single machine and a two-machine flow shop problem, by considering the possibility of delivering jobs in batches and introducing batch delivery costs.

The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs over the system while the supplier and one of the manufacturers cooperate together. The extended problem arises in the context of coordination between machine scheduling and a distribution system in a supply chain network.

The branch and bound algorithm proved to be efficient for solving the problem up to 27 jobs while the number of manufacturers and customers changes from 2 to 4. The efficiency of algorithm is based on a careful analysis of the structural properties of the problem and deriving high quality of lower bound and initial upper bound.

In addition, the reduction of total system cost in the light of cooperation between supplier and manufacturer for one sample instance was shown to be 2.69%. More discussion on the mechanism of cooperation and maximum benefit of cooperation will be presented in the next chapter.

Chapter 6

6. Benefit of cooperation

This chapter builds on the work detailed in the last three chapters to compare the computational work of various algorithm developed in the thesis and to highlight the benefit of cooperation. Furthermore the mechanisms of such cooperation will be reviewed. We first present an example that provides significant benefit from cooperation. We then consider briefly the mechanisms of cooperation and finally the result of computational experience will be presented.

Example: By presenting a parametric instance Hall and Potts have shown cooperation can provide a reduction up to 20% in total system cost. Consider the following instance, which is constructed in accordance with their example.

	Supplier		Manufacturer	
Job	1	2	1	2
Processing time	1	200	1	1
Delivery Cost	199		200	

Applying the algorithm of chapter 3 to the two jobs of supplier, leads to the optimum schedule in which the jobs are processed and delivered in two separate batches with total flow time plus delivery cost of 600. For this example, it is easy to see that supplier's processing of the jobs is completed at times 1 and 201, respectively. Then, if the jobs are to be delivered in a single batch the total flow time plus delivery cost is: 2 (1 + 200) + 199 = 601, that is increased by 1. Consequently the supplier delivers the jobs in two separate batches to manufacturer.

For the manufacturer's problem the jobs are ready at their release dates of 1 and 201 respectively. Applying the algorithm of chapter 4 to the manufacturer problem with the above processing times and release dates leads to the optimum sequence with the sum of completion times plus delivery cost of 604. Subtracting the sum of release dates from this value will yield the sum of flow times plus delivery cost of 402.

For this example it is easy to see that from the viewpoint of manufacturer it makes no difference if the jobs are delivered in a single batch or two separate batches to the customer. Hence the sum of total flow time plus delivery time over the system (STFD), i.e. for the supplier and manufacturer, before cooperation is: STFD = 600 + 402 = 1002.

At the same time applying the algorithm of chapter 5 for a combined problem to the above instance leads to the optimum schedule with total flow times and delivery cost of 805. Therefore the cooperation reduces the total system cost by 197, i.e. 19.66%.

It is worth noting that the optimum schedule for the combined problem will be obtained by the sequence in which the supplier delivers the jobs in a single batch to the manufacturer. This sequence costs the supplier more than the sequence in which the jobs are delivered to manufacturer by two batches.

6.1 Practical application

It is clear and understandable that real world scheduling problems are very different from the mathematical models and theoretical application use to study by researchers in academia. Considering these differences and making a real world application is outside the remit of this thesis. Those interested in the subject can refer to Piendo [78]. He has highlighted that "it is not easy to list all the differences between these problems and theoretical models because every real-world scheduling problem has its own particular idiosyncrasies". However, he has mentioned to a number of common differences, which are important.

In the following example we consider a practical example that helps better understanding of our works in the last three chapters. On the other hand this example provides a useful insight for developing a real world application for whose are interested this subject.

Example: Consider the following combined problem. There are 10 jobs to be scheduled by the supplier for delivery in batch to three manufacturers, say manufacturer 1, 2 and 3, with delivery costs of 504, 443 and 799 respectively as it is shown in table 6.1. Furthermore, manufacturer 1 has to process and deliver jobs to three customers, customers1, 2 and 3, with delivery costs of 288, 445 and 416 respectively as it is illustrated in table 6.2.

Table 6.1:

Problem from the viewpoint of Supplier										
	Manufacturer 1						Manufacturer 2		Manufacturer 3	
Job number	1	2	3	4	5	6	1	2	1	2
Job processing time	352	377	377	399	497	665	186	359	88	852
Delivery cost	504						443		799	

Table 6.2:

Problem from the viewpoint of manufacturer 1						
	Customer 1		Customer 2		Customer 3	
Job number	1	6	2	4	3	5
Job processing time	718	1043	792	876	810	893
Delivery cost	288		445		416	

Problem from the viewpoint of supplier:

Assuming the supplier has only one machine, the supplier faces a problem just like the one we considered in chapter 3, i.e. the problem of minimizing the sum of flow times with delivery cost on a single machine. Thus, the optimal schedule from the viewpoint of supplier can be obtained by applying the DP algorithm of Hall and Potts or the B&B algorithm provided in chapter 3. The optimal sequence is |(13)|, |(12)(22)|, (11)(21)|, |(31)(41)|, |(51)|, |(61)|, |(23)| for the sum of flow times of 18441, sum of delivery costs of 4057 and the total cost of 22498. In accordance to this sequence, the jobs will be delivered to manufacturers by the following schedule:
Job 1 of manufacturer 3 with one delivery that will be ready at time 88.
Jobs 1 and 2 of manufacturer 2 with one delivery that will be ready at time 633.
Jobs 1 and 2 of manufacturer 1 with one delivery that will be ready at time 1362.

Jobs 3 and 4 for manufacturer 1 with one delivery that will be ready at time 2138.
Job 5 of manufacturer 1 with one delivery that will be ready at time 2635.
Job 6 of manufacturer 1 with one delivery that will be ready at time 3300.
Job 2 of manufacturer 3 with one delivery that will be ready at time 4152.

Problem from the viewpoint of manufacturer 1:

Clearly, manufacturer 1 cannot start the process of jobs before the batches arrive. In fact, scheduling and batching decisions, which is made by the supplier, defines a release time for each job before which the manufacturer cannot start the processing of jobs. The optimal sequence that introduced by the supplier determines the release time of each job. Then, the problem from the viewpoint of manufacturer 1 can be illustrated as follows.

Table 6.2.a:

Problem from the viewpoint of manufacturer 1 according to time at which batches arrive						
	Customer 1		Customer 2		Customer 3	
Job number	1	6	2	4	3	5
Job processing time	718	1043	792	876	810	893
Job release time	1362	3300	1362	2138	2138	2635
Delivery cost	288		445		416	

It is worth noting that the problem illustrated in table 6.2.a can equivalently be shown in table 6.2.b. In table 6.2.b the release times are shifted such that manufacturer 1 can start the process of jobs at time zero.

Table 6.2.b:

Problem from the viewpoint of manufacturer 1 according to time at which batches arrive						
	Customer 1		Customer 2		Customer 3	
Job number	1	6	2	4	3	5
Job processing time	718	1043	792	876	810	893
Job release time	0	1938	0	776	776	1273
Delivery cost	288		445		416	

Assuming manufacturer 1 has only one machine, he faces a problem just like the one we considered in chapter 4, i.e. the problem of minimizing the sum of flow times

with delivery cost in presence of release time on a single machine. The optimal schedule from the viewpoint of manufacturer 1 can be obtained by applying the DP algorithm of Hall and Potts or the B&B algorithm provided in chapter 4. The optimal sequence is |(12)|, |(11)|, |(13)|, |(21)|, |(23)|, |(22)| for the sum of completion times of 25137 and sum of delivery costs of 2298. Subtracting the sum of release times, i.e. 12935, from the sum of completion times will yield 12202. Thus, the sum of flow times plus delivery times from the viewpoint of manufacturer is 1202 + 2298 =14500.

Therefore, the total cost of system, including the supplier and manufacturer 1, will be found by summation over the supplier's total cost (22498) and manufacturer 1's total cost (14500), which equals 36998.

The problem in the light of cooperation:

Now let us assume that the supplier and manufacturer 1 cooperate together for reducing the total system cost. In this situation they face a combined problem just like the one we considered in chapter 5. This problem can be illustrated as follows.

Table 6.3:

	Manufacturer 1						Manufacturer 2		Manufacturer 3	
J.N	1	2	3	4	5	6	1	2	1	2
Processing Time	352	377	377	399	497	665	186	359	88	852
	Manufacturer 1						Manufacturer 2		Manufacturer 3	
	Customer 1		Customer 2		Customer 3		(Virtual Customer 4)		(Virtual Customer 5)	
J.N	1	6	2	4	3	5	1	2	1	2
Processing Time	718	1043	792	876	810	893	0	0	0	0

Table 6.4:

	Manufacturer 1			Manufacturer 2	Manufacturer 3
Delivery cost	504			443	799
	Customer 1	Customer 2	Customer 3	Virtual Customer 4	Virtual Customer 5
Delivery cost	288	445	416	0	0

It is very important to note that the DP algorithm of Hall and Potts provided for this problem cannot solve this instance. As it is explained in chapter 5, the DP can only solve the problem instances with about 5 jobs and at most 2 manufacturers and 2 customers. This is while the B&B algorithm provided in chapter 5 can solve this problem instance less than 1 second. The optimal schedule will be obtained by the sequence |(11)|, |(13)|, (12)|, |(21)|, |(22)|, |(31)|, |(41)|, |(23)|, |(51)|, |(61)|, {|(11)|, |(12)|, (13)|, |(22)|, |(23)|, |(21)|} for the sum of flow times of 24495, sum of delivery costs of 7806 and the total cost of 32301 over the system.

Comparing this value with the value obtained previously, i.e. the total system cost when the supplier and manufacturer 1 had their individual sequence, proves that the total cost is reduced by 4697, which is 12.65% of total system cost.

It is worth noting this sequence requires the supplier changes his individual schedule, which was optimal. As a result the total cost from the viewpoint of the supplier will be increased to 23787, which is 1289 units greater than the last schedule. On the other hand the total cost from the viewpoint of manufacturer 1will be decreased to 8514, which shows the reduction of 5986 units.

Consequently the total cost in the light of cooperation is reduced significantly, but it is very crucial to note that the cooperation between components of a supply chain only occurs when all components feel benefit. The mechanism of cooperation such that guarantees the supplier's benefit must be offered from the manufacturer 1 (downstream) to the supplier (upstream). We will briefly consider this mechanism in the next section.

6.2 Mechanism of cooperation:

Detailed consideration of the mechanisms of cooperation between different levels of a supply chain network is outside the scope of this thesis. Hence, in this section we only address some important issues that affect the construction of such mechanisms.

Since the supplier's cost is independent of the manufacturer's decision, therefore the preliminary offer for changing the schedule cannot be proposed by the supplier. On the other hand, the manufacturer's cost is strongly dependent on the suppliers' schedule and then any change in the supplier's schedule with the gain of reducing total cost of system must be offered by the manufacturer.

Hall and Potts (2003) have suggested mechanisms by which a manufacturer may encourage a supplier to accept delivery in batches, which are more favoured to the manufacturer. These mechanisms are based on:

- Sharing the probable extra cost resulting from the change of schedule in the supplier's side.
- Compensating the full increased cost of supplier's side resulting from the change of schedule.
- Sharing the extra benefit that will be achieved overall the system resulting from the change of schedule.
- Force the supplier, for example by refusing to accept the delivery of batches after or before a certain date.

However, it is obvious that cooperation only occurs while all participants feel their involvement is beneficial. Therefore, the mechanism of cooperation may change from case to case and will be derived through negotiation and sharing of information throughout the chain.

6.3 Computational Results

To consider the benefit of cooperation between the supplier and manufacturer 1, we resorted to generating two sets of problem instances.

For each set, the number of manufacturers and customers were generated randomly from 1 to 4. The jobs were randomly distributed among manufacturers and customers, with each being assigned at least two jobs, while the number of jobs for manufacturer 1 is the sum of jobs that is assigned for customers. For the first set, the processing times and delivery times for each manufacturer and customer were randomly generated integers from the uniform distribution defined on [1, 1000].

For the second set, the processing times and delivery times for each manufacturer were randomly generated integers from the uniform distribution defined on [1, 100], while the processing times and delivery times for each customer were randomly generated integers from the uniform distribution defined on [100, 1000]. For each set, 50 problem instances were generated. Each of the two tables below shows the result for each instance when the supplier and manufacturer 1 act independently; when they cooperate together; and the comparison of the results.

As can be clearly seen in tables 1 and 2, the benefit of cooperation between supplier and manufacturer is evident in 42% of the instances of the first set and 94% of the instances of the second set. The tables also show the cooperation is not beneficial in 46% of the instances of the first set and in only 4% of the instances of the second set. The consequences of the cooperation for the remaining instances are identical. The maximum advantage gained over all instances in the two sets is 12.35% of total cost.

It is worth noting that the disadvantage of cooperation for some instances stems from the assumptions that we applied for solving the combined problem. Firstly, we assumed that each batch, which is prepared by the supplier for delivery to manufacturer 1, only contains the jobs that are destined for a particular customer. This assumption simplifies the problem solution, but it can impact on the total system cost. This is because, without this assumption, these jobs, even if they are destined for different customers, can still be delivered within the same batch to manufacturer 1. Secondly, we assumed that the two-stage jobs for each customer are sequenced in SPT order by the supplier according to the processing time of jobs only on the supplier's machine. This assumption can also impact on the total system cost while the processing time of some jobs on the second machine, which must be scheduled earlier, are greater than others that must be scheduled later.

However, close examination of table 2 reveals that the benefit of cooperation is very significant when the processing times of jobs on the second machine are greater than those on the first machine. This is particularly true when jobs are ordered in SPT on both machines.

Table 1: Comparing the sum of flow times plus delivery cost over the system before and after cooperation with processing times distribution on [1, 1000], while the jobs are randomly distributed among manufacturers and customers. Instances 1 to 25.

Supplier Flow times	Manufacturer 1 Flow times	Sum of Release times	Supplier + Manufacturer 1 Flow times	Combined Problem Flow times	Benefit of Cooperation	Benefit percent
12292	9906	6174	16024	16007	17	0.10
16398	16823	12903	20318	20318	0	0.00
27113	28953	15949	40117	37632	2485	6.19
5839	4517	2747	7609	7609	0	0.00
18509	16581	10858	24232	24696	-464	-1.91
31023	25719	21280	35462	35680	-218	-0.61
10546	11043	7108	14481	14481	0	0.00
19093	16597	8894	26796	25856	940	3.51
8325	7081	2152	13254	13810	-556	-4.19
15304	17132	10370	22066	22916	-850	-3.85
11890	13977	6518	19349	19227	122	0.63
26232	20901	15104	32029	32111	-82	-0.26
20926	23722	14416	30232	29119	1113	3.68
23019	25346	16961	31404	31387	17	0.05
20042	17448	8293	29197	28873	324	1.11
19585	13168	7938	24815	24804	11	0.04
14266	13523	4842	22947	22878	69	0.30
19912	14306	10376	23842	23842	0	0.00
22498	27435	12935	36998	32428	4570	12.35
31993	19267	10867	40393	40895	-502	-1.24
14731	17452	7215	24968	24817	151	0.60
15769	9144	4128	20785	21800	-1015	-4.88
33715	19634	12742	40607	41960	-1353	-3.33
24455	29049	21772	31732	31990	-258	-0.81
15373	22929	14141	24161	24177	-16	-0.07
17373	9072	4429	22016	22110	-94	-0.43

Table 1 continue. Instances 26 to 50

25005	12604	7536	30073	29585	488	1.62
37993	23666	17689	43970	44189	-219	-0.50
30670	23770	18057	36383	36949	-566	-1.56
30936	22439	17738	35637	35545	92	0.26
36174	15266	12285	39155	39770	-615	-1.57
26547	16287	9419	33415	33124	291	0.87
37120	26584	17801	45903	45482	421	0.92
19340	17977	10590	26727	25916	811	3.03
58672	51383	39759	70296	70587	-291	-0.41
18055	16610	11249	23416	23378	38	0.16
36311	31861	23309	44863	45720	-857	-1.91
28247	25859	15839	38267	38034	233	0.61
21850	15243	9655	27438	27784	-346	-1.26
26160	17785	10546	33399	33760	-361	-1.08
35604	13125	8756	39973	39973	0	0.00
46086	39024	31117	53993	54441	-448	-0.83
26808	16169	10918	32059	31933	126	0.39
22465	25668	8922	39211	38986	225	0.57
37725	31121	18255	50591	49859	732	1.45
30281	30432	24149	36564	36564	0	0.00
39689	27544	19872	47361	47867	-506	-1.07
55209	23308	18154	60363	60377	-14	-0.02
50958	42594	35330	58222	58310	-88	-0.15
32677	28226	19704	41199	41478	-279	-0.68

Table 2: Comparing the sum of flow times plus delivery cost over the system before and after cooperation with processing times distribution on [1, 100] for manufacturers and [100, 1000] for customers, while the jobs are randomly distributed among manufacturers and customers.

Supplier Flow times	Manufacturer 1 Flow times	Sum of Release times	Supplier+ Manufacturer 1 Flow times	Combined Problem Flow times	Benefit of Cooperation	Benefit percent
1250	2835	435	3650	3444	206	5.64
1723	4167	741	5149	4699	450	8.74
1704	3025	721	4008	3885	123	3.07
1619	5458	871	6206	5848	358	5.77
3000	4793	1408	6385	5754	631	9.88
3135	3478	1683	4930	4781	149	3.02
1441	2570	774	3237	3082	155	4.79
1821	4878	1589	5110	4936	174	3.41
3344	4873	1731	6486	6182	304	4.69
2797	5051	1751	6097	5875	222	3.64
1899	5418	1116	6201	5920	281	4.53
1010	2784	879	2915	2813	102	3.50
3147	4964	2016	6095	5807	288	4.73
3093	3335	1560	4868	4764	104	2.14
1304	3538	886	3956	3843	113	2.86
1543	3084	462	4165	3849	316	7.59
3008	5240	1525	6723	6252	471	7.01
3581	4670	1608	6643	6039	604	9.09
1225	2577	461	3341	3088	253	7.57
2918	15990	1229	17679	17189	490	2.77
1383	8882	443	9822	9491	331	3.37
2953	11104	1424	12633	12664	-31	-0.25
1156	13050	916	13290	13204	86	0.65
2129	5347	1236	6240	5828	412	6.60
1914	12299	1163	13050	13025	25	0.19
2725	9021	1315	10431	10271	160	1.53

Table 2 continue. Instances 25 to 50.

1650	3186	309	4527	4347	180	3.98
3324	13917	1819	15422	15330	92	0.60
2482	17469	1548	18403	18155	248	1.35
2454	11007	1418	12043	11726	317	2.63
3239	11226	1274	13191	12694	497	3.77
1410	11649	937	12122	12333	-211	-1.74
4646	11925	1842	14729	13978	751	5.10
2948	6054	742	8260	7951	309	3.74
2223	11733	618	13338	13082	256	1.92
1505	9092	607	9990	9742	248	2.48
3331	20661	1800	22192	21504	688	3.10
5542	16120	3127	18535	17679	856	4.62
3260	12390	2040	13610	13055	555	4.08
2411	8547	1171	9787	9787	0	0.00
3769	12770	2339	14200	14117	83	0.58
6396	16824	2727	20493	19528	965	4.71
2549	6145	310	8384	8317	67	0.80
6228	10454	1978	14704	14169	535	3.64
2553	16641	1495	17699	17316	383	2.16
3136	13925	1585	15476	14924	552	3.57
2820	12266	1286	13800	13468	332	2.41
4785	22049	1519	25315	24438	877	3.46
3674	12309	1987	13996	13472	524	3.74
1767	13521	987	14301	13979	322	2.25

6.4 Conclusion

The benefit of cooperation between the supplier and one manufacturer for reducing the total system cost, i.e. the sum of flow times plus delivery times over the system, has been briefly considered. Although the assumptions that are applied to solve the combined problem can be costly, the benefit of cooperation, especially when the processing times of jobs on the second machine are greater than of the first machine, is significant. The computational results show the reduction of total system cost of up to 12.35% while theoretical reduction of up to 20% can be achieved for special instances.

Chapter 7

7.Conclusion and further work

7.1 Conclusion

In this thesis, scheduling, batching and delivery of a set of jobs that arise in supply chain have been presented. In particular, scheduling a set of jobs in a supply chain with three stages including supplier, manufacturer and customer is analysed.

First, the problem from the viewpoint of supplier has been considered. We have provided a branch and bound algorithm for scheduling a set of jobs to be processed by the supplier on a single machine for delivery in batches to manufacturers or to other machines for further processing. This problem is a natural extension of minimizing the sum of flow times by considering the possibility of delivering jobs in batches and introducing batch delivery costs. The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs. The branch and bound algorithm proved to be very efficient. This efficiency is attributable to the sharpness of the lower bounds derived, in addition to the high quality of an initial upper bound found using an effective heuristic. Both lower bound and the upper bound were based on a careful analysis of the structural properties of the problem.

Second, the similar problem from the viewpoint of manufacturer, while the jobs are ready at their release times, has been presented. The release times were based on

arrival times of batches from the supplier. This problem is a natural extension of minimizing the sum of flow times with the general release dates by considering the possibility of delivering jobs in batches to customers. The problem in ordinary sense and even without the extension term, i.e. without considering the possibility of delivering jobs in batches, is *NP-complete*. However, it is proved that for a set of jobs under defined conditions, the sequence ordered by SPT is optimal in term of total flow time. Furthermore, it is proved that provided branch and bound algorithm for the extension version of problem is very efficient. Two alternative batch formation strategies were introduced. The first was that of continuous batching, under which the jobs that formed a batch were processed continuously; i.e., no job that belonging to another customer was processed between them and there was no idle time. The second strategy was that of discontinuous batching, where the jobs that formed a batch were processed separately but delivered together.

Third, a combined problem in the light of cooperation between the supplier and one of the manufacturers has been presented. The combined problem has been analysed as a natural extension of two problems, i.e. the problem of minimizing the sum of flow times on a single machine and a two-machine flow shop problem, by considering the possibility of delivering jobs in batches and introducing batch delivery costs for each problem. The scheduling objective adopted is that of minimizing the sum of flow times and delivery costs over the system while the system is defined as a combination of the supplier and the manufacturer that cooperate together. A two-machine scheduling problem is individually *NP-complete* and so is the combined problem. A number of propositions that dominate on the structure of optimum schedule were investigated and used to devise a branch and bound solution scheme for solving this problem. The efficiency of algorithm for solving problem instances was measured.

Fourth, the benefit of cooperation between the supplier and one of the manufacturers for reducing the total system scheduling and batching cost, i.e. the sum of flow times plus delivery costs over the system, has been investigated. The total system cost before cooperation was found by applying the algorithm provided for the first (supplier) and the second (manufacturer) problems respectively while the total system cost after cooperation was found by applying the algorithm provided for

the combined problem. Indeed it is proved that although the combined problem algorithm is provided based on some restrictive assumptions, the benefit of cooperation is still significant. The computational results showed the reduction of total system cost of up to 12.35%, while theoretically reduction of up to 20% can be achieved for special cases.

Moreover, each one of the three considered problems in the thesis has been introduced as an independent scheduling problem. From this point of view the first problem is an extension of F problem, the second problem is an extension of F problem in presence of release dates and the third one is a combined scheduling and batching problem. The efficiency of provided B&B algorithms have been compared with the existing algorithms. Indeed for each problem, it is proved B&B algorithm to be more efficient by far than the only existing algorithm, which is based on dynamic programming. This is due to the time complexity of the DP algorithm, which can be clearly observed in the escalation of computing time with the increase in the number of jobs and manufacturer (customer) for each problem.

On the other hand, the efficiency of the B&B, which enables it to solve most problem instances in less than one second for the first and the second problem and very fast for the third problem, is attributable to the tightness of the lower bound, which has been derived on the basis of careful analysis of the structural properties of each problem. This analysis provides useful insights into problems of scheduling with batching in general. In this regard, Propositions 3.1, 3.2, 3.3 and 3.8 in chapter 3, propositions 4.3.1 and 4.3.4 and Property 4.3.3 in chapter 4 and propositions 5.3.2 in chapter 5 seem most important.

In addition, effectiveness of the upper bound provided for the first (the second) problem makes it possible to use it as a fast heuristic. It is revealed that on average the upper bound heuristic produces solutions that are within 0.23% (0.33%) of the optimum.

The complexity of the dynamic programming algorithm for each one of these three problems is such that for the same number of jobs, problem instances with a larger number of manufacturers (customers) are more difficult. Interestingly, the opposite is true for the branch-and-bound algorithm, as revealed by the solution times for each problem. This is so because when the number of customers is large, the algorithm is in effect searching over a larger number of smaller subsequences,

with much smaller numbers of possible permutations. It is clearly shown that DP algorithm, even for the first problem in which the time complexity of algorithm is less than two other algorithms, cannot solve the problem instances with more than 48 jobs and with 12 manufacturers. This is while the B&B algorithm solves these problem instances very fast and even faster than problem instances with the same number of jobs and with fewer manufacturers.
Similarly, for the second problem, DP can solve the problem instances with up to 3 customers and 22 jobs or alternatively with 4 customers and 18 jobs, while the B&B algorithm proved to be efficient for solving problem instances with even more than 8 customers and 40 jobs. And finally for the combined problem, it is proved that DP cannot solve the problem instances with more than only 5 jobs, 1 (2) manufacturers and 3 (2) customers, while the B&B algorithm can solve problem instances with 27 jobs and with several manufacturers and customers efficiently.

7.2 Recommendation for further work

There are a number of issues for further investigations.

First, in the second problem we assumed that for each pair of jobs i and j, whenever $p_i \leq p_j$ then $r_i \leq r_j$. Although within the framework of supply chain management this condition may be enforced as part of the coordination between the supplier (upstream stage) and the manufacturer, however, this assumption appears restrictive. Due to the complexity of the problem it is not expected to solve the problem without any simplification, however the deterministic approaches with more generality than our work or some heuristic methods should be investigated.

Second, in the combined problem we assumed that each batch for manufacturer 1, i.e. the manufacturer who cooperates with the supplier, contains only the jobs that are destined for a particular customer. This assumption relaxed the problem to become solvable as a two-machine flowshop problem. It may be worth considering if there is still good upper and lower bound without this assumption.

Third, the algorithm provided for the combined problem consists of two sub-algorithms which are based on branch and bound method. The first algorithm

considers all of the possible combinations of delivering jobs in batch and finally provides a set of acceptable combinations of batches at leaf nodes, while the second algorithm solves a two-machine flowshop problem for each acceptable combination set of batches. It seems that the second algorithm may be improved upon by using the dominance rules already available in the literature for two-machine scheduling problem. In particular, the development of this part of the algorithm in the light of the criteria, which is presented by Della Croce, Ghirardi and Tadei [33] and Akkan and Karabati [2], should be investigated.

Fourth, the propositions and properties provided in this thesis are used for the problem of minimizing the sum of flow times plus delivery costs at different stages of a supply chain. However, some of these propositions and properties and more importantly, the logic of solution approach, are extendable for other classical scheduling problems. In particular, Hall and Potts have shown that the weighted version of this problem is intractable. But it seems that by following the logic and adopting the criteria provided in this thesis, the weighted version of the problem can be solved. This issue needs to be investigated.

Fifth, the mechanisms of cooperation suggested in this thesis are based on negotiation and sharing the information between different stages. There is a need for more accurate mechanisms of such cooperation especially, when one partner does not want to share the information with others.

Sixth, the extension of the model to multi-stage supply chain with more than three stages is another research topic.

Appendix 1

Proposition 3.13. *If the optimum set of batches established under the conditions defined in chapter 4, is sequenced in SEBT order, then either:*

A) the ready times of batches happen to be in non- decreasing order,

B) or for some subsequence, the ready times are an increasing order, in which case for any two batches x and y in the subsequence, partial schedule of the sequence $R_x + P_x \geq R_y$, *where* R *and* P *denote respectively the ready time and the total processing time.*

Proof: (By induction method). We prove this proposition in two parts. First, in part 1, the proof of proposition for the special case that the original set of jobs, i.e. the set of jobs before establishing batch between the jobs, includes only three jobs (3 single-job batches) will be presented and then, in part two, the proof will be extended to the general case.

Part 1) Assume there are 3 jobs j_1, j_2, and j_3 that may contribute for making one batch, two batches or three single-job batch together while $p_1 \leq p_2 \leq p_3$ and $r_1 \leq r_2 \leq r_3$. We consider all of the possibilities that these three jobs may make a batch.

1.1- For the case that the jobs don't contribute for making any continuous batch, i.e. the jobs establish three single-job batches, or they establish some discontinuous batches, then the statement A of the proposition is satisfied.

1.2 - j_1 and j_2 make batch b, then it is obvious that $(P_b = p_1 + p_2)/2 \leq p_3$ and $R_b \leq r_3$, so the sequence will be optimized by the *SEBT* rule and statement A, is satisfied.

1.3 - j_2 and j_3 make batch b, then it is clear that $p_1 \leq (P_b = p_2 + p_3)/2$ and $r_1 \leq R_b$, so the sequence is again optimized by the *SEBT* rule and statement A is satisfied.

1.4 - j_1 and j_3 make batch b, $P_b = p_1 + p_3$, $R_b = r_1$ or $R_b = r_3 - p_1$, $P_b/2 < p_2$ $(P_b/2 > p_2)$ and $R_b < r_2$ $(R_b > r_2)$, then the sequence will be optimized by the *SEBT* rule and statement A is satisfied.

1.5 - j_1 and j_3 make batch b, $P_b = p_1 + p_3$, $R_b = r_1$ or $R_b = r_3 - p_1$, $P_b/2 < p_2$ and $R_b > r_2$, in this case three options must be distinguished:

1.5.1) b comes first, then since $R_b + P_b \geq r_3 + p_3$ and $r_3 \geq r_2$, it is guaranteed that statement B of the proposition is satisfied.

1.5.2) j_2 comes first and $r_2 + p_2 > R_b$, then statement B of the proposition is satisfied.

1.5.3) j_2 comes first and $r_2 + p_2 < R_b$, then it requires that $r_2 + p_2 < r_3 - p_1$, or $r_2 + p_1 + p_2 < r_3$. This term means that if we start the process of job j_1 and complete the processing of job j_2, the third job, i.e. j_3, is not ready for processing yet. Consequently, the optimum sequence does not meet this option and this case making a discontinuous batch leads us to optimum schedule.

1.6- j_1 and j_3 make batch b, $P_b = p_1 + p_3$, $R_b = r_1$ or $R_b = r_3 - p_1$, $P_b/2 > p_2$ and $R_b < r_2$, in this case two options must be distinguished:

1.6.1) b comes first, then since $R_b + P_b \geq r_3 + p_3$ and $r_3 \geq r_2$, it is guaranteed that the second statement of proposition is satisfied.

1.6.2) j_2 comes first, then since $r_2 > R_b$, thus the second statement of proposition is satisfied.

Part 2) Now let us consider the possibility of adding the fourth job, j_4, to the last three jobs. It is worth noting that j_4 has the processing time of p_4, and release date of r_4, while $p_3 \leq p_4$ and $r_3 \leq r_4$.

2.1 - If p_4, is to be processed separately or all of the last three jobs have contributed for making one batch with the total processing time of $p_1 + p_2 + p_3$, then statement A is satisfied.

2.2 - If the last three jobs have not made any batch together, and j_4 may make a batch with each one of them, then three options must be distinguished:

2.2.1) j_3 and j_4 make batch b', then making such batch does not affect on the position of j_1 and j_2 and statement A is satisfied.

2.2.2) j_2 and j_4 make batch b', then making such batch does not affect on the position of j_1 and our problem will be reduced to the problem with 3 single-job batches that was considered in part 1.

2.2.3) j_1 and j_4 make batch b'. Assume that job j_3 does not exist, and then the new problem reduces to the problem of part 1. Now, if we add j_3 to the new problem three options must be distinguished (we notice that j_3 never can come before j_2):

2.2.3.1) b' comes first, followed by j_2 and j_3. It is clear that there is no confliction. Note that b' and j_2 have been considered before and satisfied the proposition.

2.2.3.2) j_2 comes first, followed by j_3 and b'. Since b' and j_2 don't conflict together and $r_3 \geq r_2$, then there is no confliction.

2.2.3.3) j_2 comes first, followed by b' and j_3. It is easy to see that there is no confliction again.

2.3 - If the last three jobs have contributed for making a batch by virtue of state 1.2, 1.3 or 1.4, then the problem again will be reduced to the problem with 3 single-job batches that was considered in part 1.

2.4- If the last three jobs have contributed for making a batch by virtue of state 1.5, 1.6 or 1.7 and job j_4 does not contribute for making a batch neither with batch b nor with single-job batch j_2, then it does not conflict with them and it must be scheduled after them.

2.5- If the last three jobs have contributed for making a batch by virtue of state 1.5, 1.6 or 1.7 and job j_4 contributes with batch b for making a bigger batch, then we have $P_{b'} = P_b + p_4$ and $R_{b'} = R_b$ or $R_{b'} = r_4 - P_b$. It is easy to see that for all these cases the logic of states 1.5, 1.6 and 1.7 is still true and the proposition will be satisfied.

2.6- if the last three jobs have contributed for making a batch by virtue of state 1.5, 1.6 or 1.7 and job j_4 contributes with single-job batch j_2 to make batch b', then we have $P_{b'} = p_2 + p_4$ and $R_{b'} = r_2$ or $R_{b'} = r_4 - p_2$. For this case five options must be distinguished:

2.6.1) $P_b/2 < P_{b'}/2$ ($P_b/2 > P_{b'}/2$) and $R_b < R_{b'}$ ($R_b > R_{b'}$), then the statement A is satisfied.

2.6.2) b comes first, $P_b/2 < P_{b'}/2$ and $R_b > R_{b'}$. Since $R_b > R_{b'}$, thus the second statement of the proposition is satisfied.

2.6.3) b' comes first, $P_b/2 < P_{b'}/2$ and $R_b > R_{b'}$. Since $R_{b'} + P_{b'} \geq r_4 + p_4$ and $r_4 > R_b$, thus the second statement of the proposition is satisfied.

2.6.4) b comes first, $P_b/2 > P_{b'}/2$ and $R_b < R_{b'}$. For this case two options must be distinguished:

2.6.4.1- $R_{b'} = r_2$. Since $R_b + P_b \geq r_3 + p_3$, and $r_3 > r_2$, thus the second part of the proposition is satisfied.

2.6.4.2- $R_{b'} = r_4 - p_2$. If $R_b + P_b \geq R_{b'}$, the second part of proposition is proved, otherwise two options must be distinguished:

2.6.4.2.1) $R_{b'} = r_2$, then it requires that $r_3 + p_3 < r_2$, which is not possible.

2.6.4.2.2) $R_{b'} = r_4 - p_2$, then it requires that $R_b + P_b < r_4 - p_2$ or $R_b + p_1 + p_2 + p_3 < r_4$. The recent term means that if we start with job j_1 and finish with job j_3, the fourth job, i.e. j_4, is not ready for processing yet. Consequently, the optimum sequence does not meet this option and in this case making a discontinuous batch leads us to optimum schedule.

2.6.5) b' comes first, $P_b/2 > P_{b'}/2$ and $R_b < R_{b'}$. Since $R_b < R_{b'}$, the second statement of the proposition is satisfied.

Carrying out with this treatment for adding jobs $j_5, j_6, \ldots,$ and j_n to the last jobs proves the proposition. It is worth noting that some jobs may don't contribute for making a batch with the previous batches or single-job batches, but they establish a batch with the jobs that have to be processed later. In this

condition the new partial schedule may be built and must be considered separately as a new partial.

References

[1] Branch and Bound. Wikipedia the free Encyclopedia . 1960.
Ref Type: Electronic Citation

[2] Akkan,C. & Karabati,S. (2004) The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm. *European Journal of Operational Research* 159, 420-429.

[3] Albers,S. & Brucker,P. (1993) The Complexity of One-Machine Batching Problems. *Discrete Applied Mathematics* 47, 87-107.

[4] Aldowaisan,T. & Allahverdi,A. (1998) Total flowtime in no-wait flowshops with separated setup times. *Computers & Operations Research* 25, 757-765.

[5] Allahverdi,A. (2000) Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times. *Computers & Operations Research* 27, 111-127.

[6] Allahverdi,A., Gupta,J.N.D., & Aldowaisan,T. (1999) A review of scheduling research involving setup considerations. *Omega-International Journal of Management Science* 27, 219-239.

[7] Arns,M., Fischer,M., Kemper,P., & Tepper,C. (2002) Supply chain modelling and its analytical evaluation. *Journal of the Operational Research Society* 53, 885-894.

[8] Baker,K. (1974) *introduction to sequencing and scheduling*. John Wiley & Sons.

[9] Ballou,R.H. (1992) *Business Logistics Management*, 3 edn. Prentice-Hall, Englewood Cliffs, NJ..

[10] Bardly,S.P., Hax,A.C., & Magnanti,T.L. (1977) *Applied Mathematical Programming*. Addison-Wesley Pub. Co.

[11] Beamon,B.M. (1998) Supply chain design and analysis:: Models and methods. *International Journal of Production Economics* 55, 281-294.

[12] Belouadah,H., Posner,M.E., & Potts,C.N. (1992) Scheduling with Release Dates on A Single-Machine to Minimize Total Weighted Completion-Time. *Discrete Applied Mathematics* 36, 213-231.

[13] Bianco,L. & Ricciardelli,S. (1982) Scheduling of A Single-Machine to Minimize Total Weighted Completion-Time Subject to Release Dates. *Naval Research Logistics* 29, 151-167.

[14] Biswas,S. & Narahari,Y. (2004) Object oriented modeling and decision support for supply chains. *European Journal of Operational Research* 153, 704-726.

[15] Budnick,F.S., McLeavey,D., & Mojena,R. (1988) *Principle of operations research for management*. Homewood, IL:Irwin.

[16] Chang,Y.C. & Lee,C.Y. (2004) Machine scheduling with job delivery coordination. *European Journal of Operational Research* 158, 470-487.

[17] Chen,I.J. & Paulraj,A. (2004) Towards a theory of supply chain management: the constructs and measurements. *Journal of Operations Management* 22, 119-150.

[18] Cheng,T.C.E., Chen,Z.L., & Oguz,C. (1994) One-Machine Batching and Sequencing of Multiple-Type Items. *Computers & Operations Research* 21, 717-721.

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

[20] Cheng,T.C.E. & Kovalyov,M.Y. (1996) Batch scheduling and common due-date assignment on a single machine. *Discrete Applied Mathematics* 70, 231-245.

[21] Cheng,T.C.E. & Kovalyov,M.Y. (2001) Single supplier scheduling for multiple deliveries. *Annals of Operations Research* 107, 51-63.

[22] Chou,C.F.M. (2004) Asymptotic performance ratio of an online algorithm for. the single machine scheduling with release dates. *IEEE Transactions on Automatic Control* 49, 772-776.

[23] Chou,F.D. & Lee,C.E. (1999) Two-machine flowshop scheduling with bicriteria problem. *Computers & Industrial Engineering* 36, 549-564.

[24] Chou,Y.-L., Pollock,S.M., Romeijn,H.E., & Smith,R.L. A Formalism for Dynamic Programming. Department of Industrial and Operations Engineering, The University of Michigan . 2001.
Ref Type: Electronic Citation

[25] Chu,C.B. (1992) Efficient Heuristics to Minimize Total Flow Time with Release Dates. *Operations Research Letters* 12, 321-330.

[26] Clausen,J. Branch and Bound Algorithms-Principles and Examples. http://www.imada.sdu.dk/~jbj/DM85/TSPtext.pdf . 1999.
Ref Type: Electronic Citation

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

[28] Conway,R.W., Maxwell,W.L., & Miller,L.W. (1967) *Theory of Scheduling*. Addison-Wesley Publishing Company, Inc..

[29] Cooper,M.C., Lambert,D.M., & Pagh,J.D. (1997) Supply Chain Management: More Than a New Name for Logistics. *The International Journal of Logistics Management* 8, 1-14.

[30] Cormen,T.H., Leiserson,C.E., Rivest,R.L., & Stein,C. (1990) *Introduction to algorithms*. Mc. Graw-Hill Company.

[31] Crauwels,H.A.J., Hariri,A.M.A., Potts,C.N., & Van Wassenhove,L.N. (1998) Branch and bound algorithms for single-machine scheduling with batch set-up times to minimize total weighted completion time. *Annals of Operations Research* 83, 59-76.

[32] Croom,S., Romano,P., & Giannakis,M. (2000) Supply chain management: an analytical framework for critical literature review. *European Journal of Purchasing & Supply Management* 6, 67-83.

[33] DellaCroce,F., Ghirardi,M., & Tadei,R. (2002) An improved branch-and-bound algorithm for the two machine total completion time flow shop problem. *European Journal of Operational Research* 139, 293-301.

[34] DellaCroce,F., Narayan,V., & Tadei,R. (1996) The two-machine total completion time flow shop problem. *European Journal of Operational Research* 90, 227-237.

[35] Deogun,J.S. (1983) On Scheduling with Ready Times to Minimize Mean Flow Time. *Computer Journal* 26, 320-328.

[36] Dessouky,M.I. & Deogun,J.S. (1981) Sequencing Jobs with Unequal Ready Times to Minimize Mean Flow Time. *Siam Journal on Computing* 10, 192-202.

[37] Dunstall,S., Wirth,A., & Baker,K. (2000) Lower bounds and, algorithms for flowtime minimization on a single machine with set-up times. *Journal of Scheduling* 3, 51-59.

[38] Dupont,L. & Dhaenens-Flipo,C. (2002) Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers & Operations Research* 29, 807-819.

[39] Dyer,M.E. & Wolsey,L.A. (1990) Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics* 26, 255-270.

[40] Elmahi,I., Merzouk,S., Grunder,O., & Elmoudni,A. A genetic algorithm approach for the batches delivery optimization in a supply chain. International Conference on Networking, Sensing & Control 1, 299-304. 2004.
Ref Type: Conference Proceeding

[41] Erenguc,S.S., Simpson,N.C., & Vakharia,A.J. (1999) Integrated production/distribution planning in supply chains: An invited review. *European Journal of Operational Research* 115, 219-236.

[42] Ganeshan,R. & Harrison,T.P. An Introduction to Supply Chain Management. http://lcm.csa.iisc.ernet.in/scm/supply_chain_intro.html . 1995.
Ref Type: Electronic Citation

[43] Garey,M.R., Johnson,D.S., & Sethi,R. (1976) The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* 1, 117-129.

[44] Goemans,M.X., Queyranne,M., Schulz,A.S., Skutella,M., & Wang,Y.G. (2002) Single machine scheduling with release dates. *Siam Journal on Discrete Mathematics* 15, 165-192.

[45] Gonzalez,T. & Sahni,S. (1978) Flowshop and jobshop scheduling: Complexity and approximation. *Operations Research* 26, 36-52.

[46] Graham,R.L., Lawler,E.L., Lenstra,J.K., & Rinnooy Kan,A.H.G. (1979) Optimization and approximation in deterministic sequencing and scheduling: A survay. *Annals of Discrete Mathematics* 5, 287-326.

[47] Gupta,J.N.D., Neppalli,V.R., & Werner,F. (2001) Minimizing total flow time in a two-machine flowshop problem with minimum makespan. *International Journal of Production Economics* 69, 323-338.

[48] Hall,L.A., Schulz,A.S., Shmoys,D.B., & Wein,J. (1997) Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research* 22, 513-544.

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

[50] Hammer,M. Reengineering the Supply Chain: An Interview With Michael Hammer. Supply Chain Management Review . 1999.
Ref Type: Electronic Citation

[51] Handfield,R.B. & Nichols,JR.E.L. (1999) *Introduction to Supply Chain Management*. Prentice-Hall, Inc..

[52] Hariri,A.M.A. & Potts,C.N. (1983) An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics* 5, 99-109.

[53] Harvey,M.G. & Richey,R.G. (2001) Global supply chain management: The selection of globally competent managers. *Journal of International Management* 7, 105-128.

[54] Hiller,F.S. & Liberman,G.J. (1995) *Introduction to operations research*, 6 edn. McGraw-Hill.

[55] Hoogeveen,J.A. & Kawaguchi,T. (1999) Minimizing total completion time in a two-machine flowshop: Analysis of special cases. *Mathematics of Operations Research* 24, 887-910.

[56] Huan,S.H., Sheoran,S.K., & Wang,G. (2004) a review and analysis of supply chain operations reference (SCOR) model. *Supply Chain Management: An International Journal* 9, 23-29.

[57] Hurink,J. & Knust,S. (2001) Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics* 112, 199-216.

[58] Ignall,E. & Schrage,L. (1965) Application of the branch and bound technique to some flow-shop scheduling problems. *Operations Research* 13, 400-412.

[59] Johnson,M.E. & Pyke,D.F. Supply Chain Management. http://mba.tuck.dartmouth.edu/pages/faculty/dave.pyke/case_studies/supply_chain_or_ms.pdf . 1999.
Ref Type: Electronic Citation

[60] Kaminsky,P. & Simchi-Levi,D. (2001) Asymptotic analysis of an on-line algorithm for the single machine completion time problem with release dates. *Operations Research Letters* 29, 141-148.

[61] Kellerer,H., Tautenhahn,T., & Woeginger,G.J. (1999) Approximability and nonapproximability results for minimizing total flow time on a single machine. *Siam Journal on Computing* 28, 1155-1166.

[62] Krajewski,L. & Wei,J.C. (2005) The value of production schedule integration in supply chains. *Decision Science* 32, 601-634.

[63] Lambert,D.M. & Cooper,M.C. (2000) Issues in supply chain management. *Industrial Marketing Management* 29, 65-83.

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

[65] Lee,Y.H. & Kim,S.H. (2002) Production-distribution planning in supply chain considering capacity constraints. *Computers & Industrial Engineering* 43, 169-190.

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

[67] Liaee,M.M. & Emmons,H. (1997) Scheduling families of jobs with setup times. *International Journal of Production Economics* 51, 165-176.

[68] Lummus,R.R. & Vokurka,R.J. (1999) Defining supply chain management: A historical perspective and practical guidelines. *Industrial Management and Data Systems* 11-17.

[69] Mao,W. & Kincaid,R.K. (1994) A look-ahead heuristic for scheduling jobs with release dates on a single machine. *Computers & Operations Research* 21, 1041-1050.

[70] Mason,A.J. & Anderson,E.J. (1991) Minimizing Flow Time on A Single-Machine with Job Classes and Setup Times. *Naval Research Logistics* 38, 333-350.

[71] Min,H. & Zhou,G.G. (2002) Supply chain modeling: past, present and future. *Computers & Industrial Engineering* 43, 231-249.

[72] Minoux,M. (1986) *Mathematical Programming Theory and algorithms*. John Wiley & Sons Ltd..

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

[74] New,S.J. (1997) The scope of supply chain management. *Supply Chain Management* 2, 15-22.

[75] New,S.J. & Payne,P. (1995) Research frameworks in logistics:three models, seven dinners and a survay. *International Journal of Production Economics* 25, 60-77.

[76] Ng,C.T., Cheng,T.C.E., Yuan,J.J., & Liu,Z.H. (2003) On the single machine serial batching scheduling problem to minimize total completion time with precedence constraints, release dates and identical processing times. *Operations Research Letters* 31, 323-326.

[77] Phillips,C., Stein,C., & Wein,J. (1998) Minimizing average completion time in the presence of release dates. *Mathematical Programming* 82, 199-223.

[78] Pinedo,M. (1995) *Scheduling Theory, Algorithms, and systems*. Prentice-Hall. Inc..

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

[80] Rinnooy Kan,A.H.G. (1976) *Machine Scheduling Problem: Classification, Complexity and Computations*. Nijhoff, The Hague.

[81] Sarmiento,A.M. & Nagi,R. (1999) A review of integrated analysis of production-distribution systems. *Iie Transactions* 31, 1061-1074.

[82] Shapiro,J.F. (2001) *Modeling the Supply Chain*. Wadsworth Group.

[83] Shapiro,J.F. Bottom-up vs. top-down approaches to supply chain management and modeling. Sloan School og Management . 1998.
Ref Type: Electronic Citation

[84] Silver,E.D. (1981) Operations research in inventory management: A review and critique. *Operations Research* 29, 628-645.

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

[86] Tan,K.C. (2001) A framework of supply chain management literature. *European Journal of Purchasing & Supply Management* 7, 39-48.

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

[88] Van De Velde,S.l. (1990) Minimizing the sum of job completion times in the two-machine flow-shop by Lagrangean relaxation. *Annals of Operations Research* 26, 257-268.

[89] Vidal,C.J. & Goetschalckx,M. (1997) Strategic production-distribution models: A critical review with emphasis on global supply chain models. *European Journal of Operational Research* 98, 1-18.

[90] Waller,D.L. (2003) *Operations Management a supply chain approach*, 2 edn.

[91] Wang,X. & Cheng,T.C.E. (2005) Two-machine flowshop scheduling with job class setups to minimize total flowtime. *Computers & Operations Research* 32, 2751-2770.

[92] Webster,S. & Baker,K.R. (1995) Scheduling Groups of Jobs on A Single-Machine. *Operations Research* 43, 692-703.

[93] Weng,Z.K. Pricing and ordering strategies in manufacturing and distribution alliances. IIE Transactions (Institute of Industrial Engineers) 29, 681-692. 1997.
Ref Type: Electronic Citation

[94] Weng,Z.K. (1999) The Power of coordinated decisions for short-life-cycle products in a manufacturing and distribution supply chain. *IEEE Transactions on Automatic Control* 31, 1037-1049.

[95] Wilf,H.S. (1994) *Algorithms and Complexity*, Internet Edition edn.

[96] Zipkin,P.H. (2000) *Foundations of inventory management*. New York, NY: McGraw-Hill.