Integrating purchase and production planning : using local search in supply chain optimization

Integrating Purchase and Production Planning

Using Local Search in Supply Chain Optimization

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

De Bontridder, Koen Margerite Jozef

Integrating purchase and production planning :
using local search in supply chain optimization /
by Koen Margerite Jozef De Bontridder. -
Eindhoven: Technische Universiteit Eindhoven, 2001.
Proefschrift. - ISBN 90-386-0961-2
NUGI 811
Subject headings: operations research; production planning / scheduling /
approximation algorithms / supply chain management
2000 Mathematics Subject Classification: 90B30, 90B35, 68W25, 90C59

Painting on the cover by Gilbert De Bontridder (Rustplaats in mijn hoofd, 1992)
Printed by Universiteitsdrukkerij Technische Universiteit Eindhoven

Integrating Purchase and Production Planning

Using Local Search in Supply Chain Optimization

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof.dr. R.A. van Santen, voor
een commissie aangewezen door het College
voor Promoties in het openbaar te verdedigen op
woensdag 21 november 2001 om 16.00 uur

door

Koen Margerite Jozef De Bontridder

geboren te Heel en Panheel

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.K. Lenstra
en
prof.dr. E.H.L. Aarts.

Copromotor:
dr. J.A. Hoogeveen.

Contents

1

Introduction

The past decade shows a growing interest in building models and algorithms for planning and scheduling problems that arise in the supply chain. Motivated by the globalization of supply chains there is a growing need for solving problems in an integrated context, which is possible nowadays, because of the advances in information systems like enterprise resource planning (ERP) and the growth of computational power. The goal of supply chain scheduling is realizing the benefits that arise from coordinated decision making.

In this thesis we deal with an example of a supply chain scheduling problem: we integrate a purchase and a production planning process. The resulting problem is handled through extensions of optimization techniques for more classical production planning problems. Most of our algorithms are based on local search techniques. In all our local search algorithms we use mathematical programming techniques to incorporate problem-specific knowledge.

This introductory chapter continues with a description of supply chain scheduling in Section 1.1. Next, we describe our supply chain scheduling problem in Section 1.2. In Section 1.3 we give a short overview of optimization techniques for production planning, and in Section 1.4 we give a short introduction into local search techniques. This chapter concludes with an overview of the thesis in Section 1.5.

1.1 Supply chain scheduling

Supply chain scheduling is the process of making a plan such that goods are produced and distributed in the right quantities, to the right locations, and at the right time in such a way that the total costs are minimized. An introduction into supply chain concepts is given by Simchi-Levi et al. [2000]. For a good description of currently available planning software like advanced planning systems (APS) we refer to Stadler and Kilger [2000]. Before we describe our supply chain scheduling problem, we give two examples. In both examples two consecutive stages of the supply chain are considered. In the first example the two consecutive stages are production and distribution. In the second example two consecutive production stages are considered.

Example 1.1. *Newspaper production and distribution planning*
Newspapers must be printed as late as possible in order to include the last minute news, but they have to be delivered at the homes of subscribers before a given deadline. The time available for printing and distributing the newspapers is therefore limited. Hence, the production and the distribution must be done in a very efficient way. Teeuwen [1999] describes an approach to integrate the production and distribution planning of two regional daily newspapers in the Netherlands.

Both newspapers have several editions, which are printed at one location. At this location four identical presses are available. Before an edition can be printed, printing plates have to be installed on the press. The time at which the printing plates for an edition become available differs per edition. Between the printing of two different editions some of the printing plates must be replaced. The time required for this depends on the difference between the two editions.

After newspapers have been printed they are loaded into vehicles and transferred to drop-off points. From there they are distributed to the subscribers. The newspapers that must be delivered at a given drop-off point are carried by the same vehicle, and in case they belong to the same edition they must be printed without interruption. Between the printing and the departure of the vehicle only a maximum period of time may pass as a consequence of limited storage capacity. The goal is to find a production and distribution plan such that the costs are minimized, where the costs depend on the number and the types of vehicles to be used. □

Example 1.2. *The integration of two production stages in the steel industry*
This supply chain scheduling problem deals with the production process of coils, which proceeds in two stages. In the first stage liquid steel is cast into a solid steel band using a *continuous caster* and the resulting band is then cut into slabs. In the second stage the slabs are rolled into coils in the *hot strip mill*. Traditionally these

two production processes are decoupled by allowing a large inventory of slabs between the two production stages. This is useful since both planning problems have rather different objectives. The production plan for the continuous caster is determined by the continuous arrival of liquid steel that must be casted immediately, and the production plan for the hot strip mill is determined by the demand of the customers. Cowling and Rezig [2000] describe an approach to integrate the two individual planning problems in order to increase the profit by achieving a higher customer satisfaction, a quality improvement of the coils, and a reduction in production costs.

The production costs can be reduced by realizing smaller stocks of slabs between the two production stages. Furthermore, significant energy savings can be realized when the slabs are hot-charged directly form the continuous caster into the second production stage. Slabs that are used in the hot strip mill to produce a specific customer order must have a certain quality. A measurement for the quality of the slabs is their grade. During the casting batches of slabs with a single grade are produced. These batches can weigh 300 tonnes. On the other hand there are customer orders of only a few tonnes. Therefore, a good coordination between the produced slabs and the customer demand is needed in order to realize smaller stocks. To simplify the coordination steel of a quality higher than required can be delivered to the customer. By upgrading the quality of an order larger batches of coils with the same grade can be realized. Hereby one must take into account that upgrading the quality of an order may reduce the profit. □

1.2 Problem under consideration

We look at supply chains that involve a manufacturer producing items for customers. Each item should be produced before a given due date. The manufacturer is penalized if the items are completed too late. The penalties depend linearly on the size of the delay. We refer to all costs due to violations of the due dates as tardiness costs.

The production process of each item consists of one or more steps. Such a step in a production process is called an operation. In the *production plan* we describe for all operations the starting times. An operation must be processed on a given machine, which can handle at most one operation at a time. The limited capacity of the machines is one of the main restrictions that are to be considered during the design of a production plan. Another important restriction is the limited availability of material that is consumed in the production of goods. We refer to these materials as *nonrenewable resources*. An operation can only start if all required 'nonrenewables' are available. All nonrenewables must be produced by other operations or

purchased from external suppliers.

In the *purchase plan* we describe the suppliers, the purchased quantities, and the moments of receipt of all purchased nonrenewables. To design a purchase plan we must consider several suppliers. All of these suppliers have their own prices, discounts, and restrictions. In such an environment, the manufacturer can save money by grouping orders for nonrenewables. In order not to delay the production plan, the only option is to advance orders. However, such decisions increase the inventory costs of the nonrenewables.

The goal of the manufacturer is to ensure that his purchase and production plan is designed in such a way that the costs are minimized. These costs consist of purchase, tardiness, and inventory costs. We use an example to describe our supply chain scheduling problem in more detail. In this example we consider a manufacturer who produces pre-stretched canvases, which are used as painting surfaces. More complicated examples of our problem can be found in the semi-conductor and the computer industry.

Example 1.3. *The production of pre-stretched canvases*
To produce a pre-stretched canvas four stretcher strips and a piece of cotton canvas are required. The stretcher strips, which are made of fir wood, are purchased from external suppliers or produced by the manufacturer himself. The fir wood and the cotton canvas are always purchased from external suppliers. To produce the pre-stretched canvases the manufacturer has three workstations. At the first workstation there is one sawing machine to make stretcher strips from fir wood. At the second workstation frames are constructed by putting the four stretcher strips together. At the third workstation a tacker is used to stretch the cotton canvas over the frame. In this example we make no distinction between stretcher strips of different sizes and we assume that the three workstations are continuously available.

The production process of pre-stretched canvases consists of two operations For each operation a release time at which it becomes available, a machine that has to process it, and a positive processing time are given. In our example the workstations are the machines. The processing time for the construction of one frame on the second workstation is 10 minutes, and processing time for the stretching of one canvas is 15 minutes. In our example there are no release times.

The relative ordering of the operations is specified by a *precedence relation*. If a precedence constraint between two operations u and v exists, then operation v cannot start before the completion of operation u. The precedence relation of the production process of pre-stretched canvases is depicted in Figure 1.1. The operations are represented by rectangles, whose widths correspond to the processing times. The nonrenewables and the delivered items are represented by diamonds,

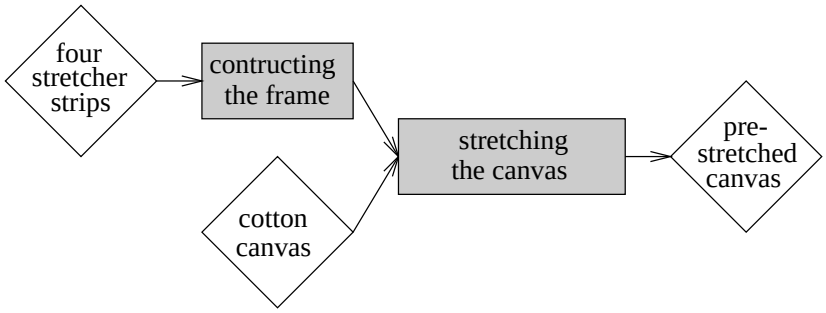

Figure 1.1: The production of pre-stretched canvases.

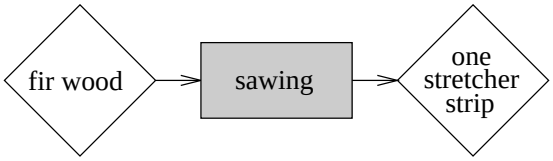

Figure 1.2: The production of stretcher strips.

and the precedence relations are represented by arcs. In our model we allow all acyclic routings. Some precedence constraints may also stipulate a *positive end-start time lag*. This means that between the completion of the first operation and the start of the second operation a given positive amount of time must elapse. In our example there are no time lags.

Fir wood, cotton canvases and stretcher strips are examples of nonrenewables. For each operation a list of required nonrenewables is given. The operation can only start when all required nonrenewables are available in sufficient quantities. Nonrenewables must be produced by other operations or purchased from external suppliers. One stretcher strip can for instance be produced in 10 minutes on the first workstation. The routing to produce the stretcher strips is depicted in Figure 1.2. A nonrenewable that is produced by another operation is only available for consumption at a given amount of time after the completion of the producing operation. In our example the stretcher strips are available immediately after the completion of the sawing.

The cotton canvas and the stretcher strips can be purchased from two suppliers. For the fir wood only one supplier is available. All of these suppliers have their own prices, discounts, and other restrictions. Nonrenewables can only be purchased at given moments in time, i.e., at the beginning of the week. The purchased quantity of a nonrenewable from a supplier must be either zero or between a given lower and

upper bound. Furthermore, we assume that for each nonrenewable the total quantity that is to be purchased is given.

The manufacturer has to assign all operations to a starting time and all purchased nonrenewables to a supplier and to a moment in time in such a way that all constraints are satisfied and the sum of all costs is minimized. As mentioned before the costs consist of purchase, tardiness, and inventory costs.

The *purchase costs* are the true purchase costs minus the discounts. The true cost function for purchasing a quantity of a nonrenewable at a moment in time from a supplier is piecewise linear, strictly increasing, and concave. In addition the manufacturer receives discounts based on the total quantity bought from one supplier. Discounts are always based on a weighted sum of the purchased quantities: we assume that the discount function is piecewise linear, convex, and nondecreasing.

The *tardiness costs* are calculated on the basis of a subset of the operations, the so-called end-operations, which are the operations that produce the items that must be delivered to the customers. For each end-operation a due date and a positive weight are given. The *lateness* of an end-operation is defined as its completion time minus its due date. The *tardiness* of an end-operation is defined as the maximum of its lateness and zero. The *tardiness costs* are equal to the weighted sum of the tardinesses of all end-operations.

The *inventory costs* are the costs of holding purchased nonrenewables during the production process. A purchased nonrenewable stays in the production process from the moment of receipt until the moment that the resulting end item is delivered to the customer. An item is delivered to a customer at the maximum of the completion time and the due date of the corresponding end-operation. A part of the inventory costs are included in the tardiness costs. Therefore, we use in the remainder of the thesis the term inventory costs to denote the costs from the moment of receipt until the due date of the resulting item. These inventory costs are equal to a weighted sum of the due dates minus a weighted sum of the moments of receipt of the purchased nonrenewables, where the weights are based on the costs of holding one unit of inventory of the purchased nonrenewable for one time unit.

Figure 1.3 shows a production plan for six pre-stretched frames with a due date of 80 minutes. The depicted blocks correspond to the processing of the operations. The operations with the same color (grey scale) depict the production plan of each of the pre-stretched canvases. In this example 8 stretcher strips are produced on workstation 1 and 16 stretcher strips are purchased from external suppliers. The assignment of the purchased nonrenewables to a supplier and a moment in time is part of our problem. Nonrenewables can only be purchased at the beginning of each hour. All purchased stretcher strips, three pieces of cotton canvas, and all fir wood are purchased at time zero. The rest of the cotton canvases are purchased after 60

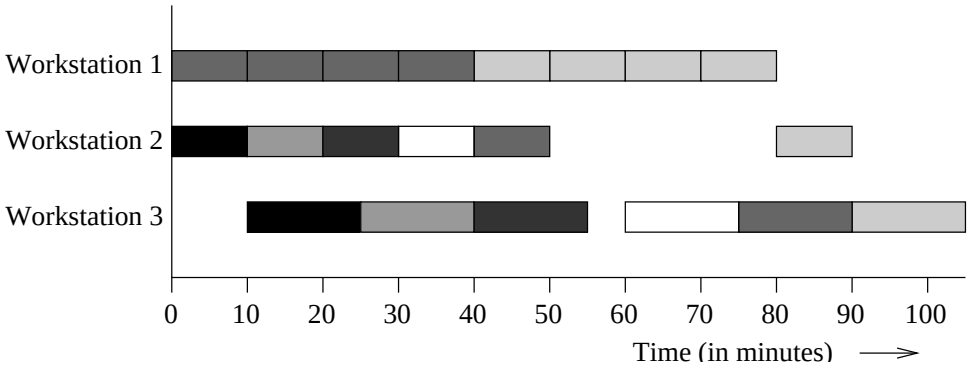

Figure 1.3: A production plan.

minutes. Let the costs for holding one piece of fir wood, one stretcher strip, and one piece of cotton canvas in inventory for one minute be € 0.02, € 0.04, and € 0.02, respectively, and let the costs for a violation of a due date be € 0.25 per minute. The inventory costs for this example are calculated as follows. The three pre-stretched canvases that are completed in the first 60 minutes are made of four purchased stretcher strips and one piece of cotton each. All of these nonrenewables are present in the production process from the first until the 80th minute. The inventory costs of one such a pre-stretched canvas is € $4 \cdot 0.04\ (80-0) + 0.02\ (80-0) =$ € 14.40. In the same way we can calculate the inventory costs for the other pre-stretched canvases: for the pre-stretched canvas completed after 75 minutes the inventory costs are € $4 \cdot 0.04\ (80-0) + 0.02\ (80-60) =$ € 13.20, and for the other two pre-stretched canvases the inventory costs are € $4 \cdot 0.02\ (80-0) + 0.02\ (80-60) =$ € 6.80. So the total inventory costs are € 70.00. Furthermore, two pre-stretched canvases are completed too late. The resulting tardiness costs are € $(90-80)\ 0.25 + (105-80)\ 0.25 =$ € 8.75. We do not discuss the purchase costs for this example. □

1.3 Optimization techniques for production planning

Scheduling is the process of allocating scarce resources to activities over time. In the scheduling literature a lot of attention has been paid to problems that arise in production planning. In these models the scarce resources are the machines, and the activities are the operations that have to be performed on these machines. Other applications of scheduling models can be found in computer control, personnel planning, and maintenance scheduling. Lawler et al. [1993] give a review of the main contributions to scheduling. This review is continued by Hoogeveen et al. [1997]. For an introduction to scheduling we refer to Pinedo [1995].

The basic model for many practical production planning problems is the *job shop scheduling problem*.

Definition 1.1. *In the job shop problem we have to schedule a set of jobs on a set of machines subject to the following constraints. Each job consists of a number of operations, which have to be processed in a given order, each on a specified machine. Each machine can only handle one operation at a time. For all operations a processing time is given. It is not allowed to interrupt the processing of an operation. Subject to these constraints, we want to find a schedule for which the completion time of the last operation (makespan) is minimal.*

Like many other scheduling problems the job shop scheduling problem belongs to the class of *NP*-hard problems; see Garey and Johnson [1979]. It is generally believed that all instances of *NP*-hard problems cannot be solved to optimality within an amount of time that is polynomially bounded by the size of the problem instance. If we want to be certain to find an optimal solution for the job shop scheduling problem we can apply an enumerative algorithm like *branch and bound*. Despite the progress that has been made in the last years, enumerative algorithms still need enormous amounts of computation time to solve problem instances of a realistic size. Consequently, there is a lot of interest in approximation algorithms that can find near-optimal solutions in reasonable computation times. Roughly speaking there are two broad classes of approximation algorithms: constructive methods and local search methods. *Constructive methods* create solutions by using some specific rules. Examples of such an approach are list scheduling, linear programming and rounding, and constraint propagation algorithms.

Local search methods employ the idea that a given solution may be improved by making small changes. In the next section we give a short introduction into local search techniques. There are also various solution methods that combine local search techniques with constructive and enumerative methods. A well known algorithm that combines constructive methods with local search is the shifting bottleneck algorithm for the job shop scheduling problem. In this method the local search algorithm is guided by construction algorithms that generates solutions to single machine scheduling problems. An overview of local search algorithms for the job shop scheduling problem is given by Vaessens et al. [1996].

In the job shop scheduling problem the machines and the operations are of a relatively simply nature. The operations only require one machine, and the machines are only able to process at most one operation at a time. It is obvious that in many practical situations more degrees of freedom are required, i.e., operations may require several machines simultaneously or machines may be able to process more than one operation at a time. These kinds of extensions and many others are

covered by the area of *resource-constrained project scheduling*. Extensive reviews are given by Herroelen et al. [1998] and Brucker et al. [1999].

Another important extension of the job shop scheduling problem is *flexible job shop scheduling*. In the flexible job shop problem operations can be processed by any machine from a given set. The problem is to assign each operation to a machine and to order the operations on the machines, such that the makespan is minimized. An example of a local search algorithm for the flexible job shop scheduling problem is given by Mastrolilli and Gambardella [2000].

1.4 Local search techniques

In most of our algorithms we use local search techniques. We refer to Aarts and Lenstra [1997] for an overview of local search. In this section we give some guidelines for applying local search techniques. We use the job shop scheduling problem to illustrate some concepts.

Local search is based on the idea that a given solution may be improved by making small changes. Such a small change is called a move. The set of solutions that can be reached from a given solution by making one move is called its neighborhood. Roughly speaking a local search algorithm starts with an initial solution and then continually tries to find better solutions by selecting in each iteration a neighbor of the current solution. To apply a local search algorithm to a specific problem the following three questions must be taken into consideration.

- How to navigate through the solution space?
- How to incorporate problem-specific knowledge?
- How to control the computational effort?

Below we elaborate on each of these items in more detail. A basic method to navigate through the search method is *iterative improvement*. Iterative improvement starts from an initial solution and repeatedly selects neighbors as long as they improve on the current solution. Consequently, the algorithm stops when it reaches a local minimum.

Escaping from poor local minima is one of the main challenges of local search. Another challenge is to avoid *cycling*, i.e., revisiting the same solution over and over again. Three well-known search strategies that try to handle these challenges are *simulated annealing*, *tabu search*, and *variable depth search*.

Simulated annealing was introduced by Kirkpatrick et al. [1983] and Černý [1985]. In each iteration the algorithm evaluates the neighbors in a random order until a neighbor is accepted. In addition to the better cost neighbors, which are always accepted, the algorithm also accepts worse neighbors with a probability

that is gradually decreased in the course of the algorithm's execution. Decreasing the acceptance probability is controlled by a set of parameters whose values are determined by a cooling schedule.

Tabu search algorithms were introduced by Glover [1989, 1990]. In each iteration a tabu search algorithm forbids certain moves. Such a forbidden move is called tabu. The set of moves that are tabu is often stored in so-called 'tabu lists'. These lists are dynamically updated during the execution of the algorithm. The tabu search algorithm always accepts the best neighbor that is not tabu.

Variable depth search, which was introduced by Kernighan and Lin [1970] starts from an initial solution and generates a sequence of subsequent neighbors by making relatively small changes. From this sequence one solution is selected to serve as the initial solution for a next sequence. During the generation of a sequence tabu search techniques are often incorporated to prevent cycling. All kinds of variants of these search methods have been proposed in the literature.

Problem-specific knowledge may be incorporated in the choice of a *representation method* and a neighborhood. The representation of a solution must be chosen in such a way that all relevant solutions and suitable neighbors can be represented, but also in such a way that the solution space is small. Another important feature of a good representation method is that the objective value of the current solution and all of its neighbors can be calculated easily. Therefore, the choice of the representation depends on the choice of the neighborhood. A good example of a representation method is the disjunctive graph representation of Roy and Sussmann [1964] for the job shop scheduling problem. We use the following example for the job shop scheduling problem to illustrate this representation method.

Example 1.4. In our example we have 3 machines, 3 jobs, and 9 operations. The orders for jobs A, B, and C are $1 \rightarrow 2 \rightarrow 3$, $4 \rightarrow 5 \rightarrow 6$, and $7 \rightarrow 8 \rightarrow 9$, respectively. In the table below we give for each operation u the required machine m_u and the processing time p_u.

u	1	2	3	4	5	6	7	8	9
m_u	1	2	3	2	3	1	2	1	3
p_u	9	8	2	4	2	4	3	6	7

In Figure 1.4 we give an example of a solution for this problem. In this solution the processing orders are $1 \rightarrow 6 \rightarrow 8$ on machine 1, $7 \rightarrow 4 \rightarrow 2$ on machine 2, and $5 \rightarrow 9 \rightarrow 3$ on machine 3. □

The solution depicted in Figure 1.4 is an example of a so-called left-justified schedule.

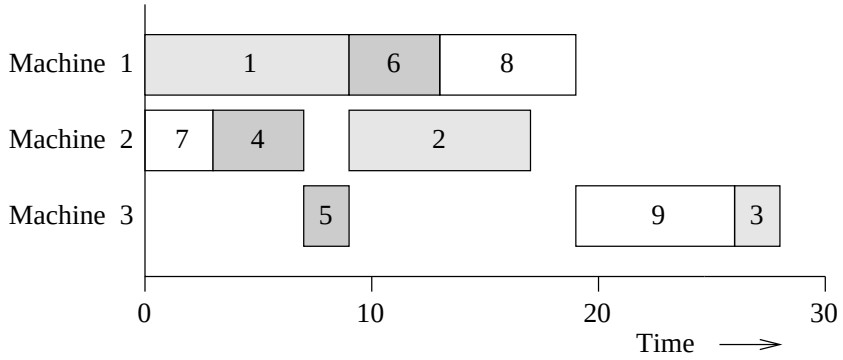

Figure 1.4: A solution.

Definition 1.2. *A feasible schedule is called left-justified if no operation can start earlier without*

- *delaying another operation, or*
- *changing the processing order on any machine.*

In standard job shop scheduling the first restriction is a direct consequence of the second restriction. Now we can make the following trivial observation.

Observation 1.3. *For the job shop scheduling problem there is always an optimal schedule that is left-justified.* □

Apparently we can restrict our attention to the set of left-justified schedules.

An instance for the job shop scheduling problem can be represented by a *disjunctive graph*. In the *disjunctive graph* each operation is represented by a node, whose weight is equal to the processing time of the operation, and each precedence relation is represented by an arc. If two operations are processed on the same machine, then the two corresponding nodes are connected by an edge. Given the processing order on each machine, we can orient these edges. The corresponding digraph represents a left-justified schedule: the makespan of a solution is equal to the longest path. Balas [1969] proved that reversing an oriented edge on a longest path in the digraph always results in a feasible schedule.

Example 1.5. In Figure 1.5 we give an example of a disjunctive graph and the digraph for the solution depicted in Figure 1.4. In the digraph we left out some redundant arcs and we included a longest path. The makespan of the solution depicted is 28. □

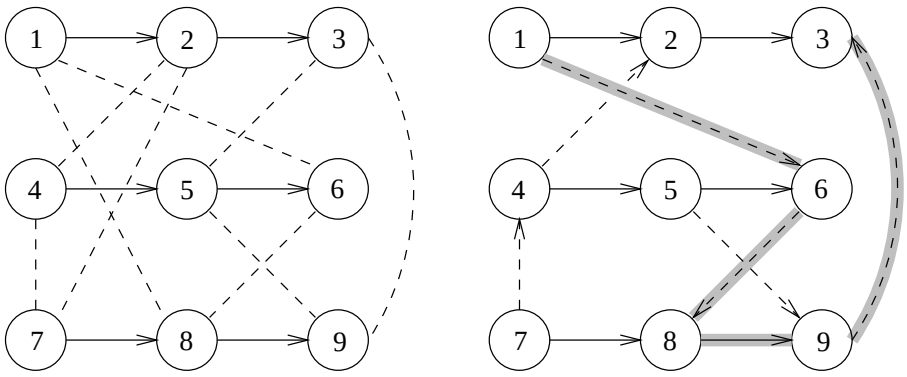

Figure 1.5: The disjunctive graph and the digraph for the solution depicted in Figure 1.4.

The neighborhood for each solution is defined by a mapping, the so-called *neighborhood function*. A neighborhood function imposes a directed graph on the solution space. The nodes of this *neighborhood graph* are the solutions, and there is an arc from a node to another node if the latter node is a neighbor of the former one. A theoretical measurement for the possibility of reaching solutions is the connectivity of the neighborhood function.

Definition 1.4. *A neighborhood function $\mathcal{N}$ is called strongly connected if the corresponding neighborhood graph is strongly connected. A neighborhood function $\mathcal{N}$ is called optimum connected if for each solution there exists a path to an optimal solution in the corresponding neighborhood graph.*

In practice connectivity is not a necessary condition for a successful local search algorithm as we will see later on. By using good initial solutions or restarting mechanisms we can overcome a lack of connectivity. To select appropriate initial solutions and restarting mechanisms we can use problem-specific knowledge and information gathered during earlier parts of the search process. Hereby we must take into account that it is important to search promising regions in more detail.

The computational effort of searching a neighborhood could be reduced by using a first improvement strategy or by estimating the objective values of the neighbors. In a first improvement search strategy we accept the first neighbor that improves upon the current solution. With a first improvement search strategy the order in which the neighbors are evaluated affects the search process. Evaluating the neighbors in a random order is useful to prevent cycling. If computing the exact objective value takes too much time then the value of neighbors can be estimated by using strong lower bounds. An example of such an approach for the job shop

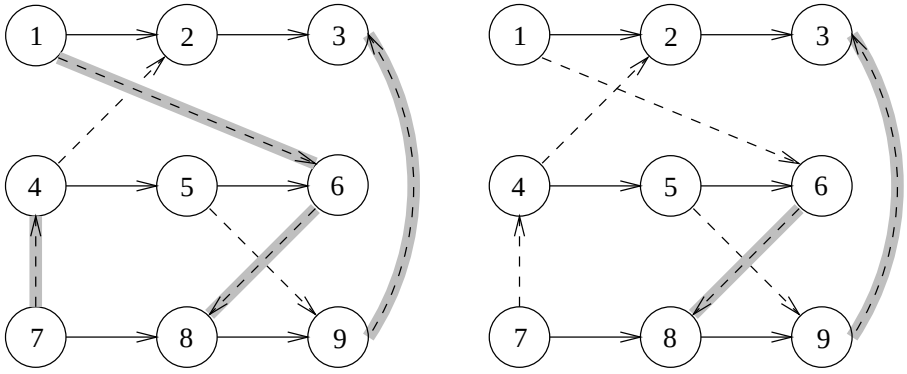

Figure 1.6: The oriented edges that must be reversed to obtain the neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively.

scheduling problem is given by Taillard [1994].

Now we discuss two neighborhood functions for the job shop scheduling problem. Both neighborhood functions are based on the disjunctive graph representation, which has been described above. Van Laarhoven et al. [1992] propose a neighborhood function $\mathcal{N}_1$, which obtains a neighbor by reversing an oriented edge on any longest path. They prove that the resulting neighborhood function is optimum connected.

Nowicki and Smutnicki [1996] propose a neighborhood function $\mathcal{N}_2$ that is not optimum connected. To define their neighborhood function they consider only a single arbitrarily selected critical path and use the concept of *blocks*. A block is a maximal sequence of adjacent operations that are processed on the same machine in a chosen longest path. Each neighborhood consists of all orientations obtained by swapping the first two or the last two operations of each block consisting of at least two operations. In the first block only swaps of the last two operations are considered, and in the last block only swaps of the first two operations are considered. They prove that neighbors that are in $\mathcal{N}_1 \setminus \mathcal{N}_2$ will never improve the current solution. Furthermore, they prove that an empty neighborhood implies optimality. Computational experiments show that local search algorithms based on neighborhood function $\mathcal{N}_2$ are more efficient than local search algorithms based on neighborhood function $\mathcal{N}_1$.

Example 1.6. For the example depicted in Figure 1.5 the neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ consist of four and two solutions, respectively. All of these solution are obtained by reversing one oriented edge. The oriented edges that must be reversed to obtain such a neighbor are highlighted in Figure 1.6: for neighborhood $\mathcal{N}_1$ in the digraph on the left and for neighborhood $\mathcal{N}_2$ in the digraph on the right. □

1.5 Overview of the thesis

This thesis is organized as follows. In the following three chapters we consider three subproblems of the supply chain scheduling problem described in Section 1.2. In Chapter 5 we integrate these algorithms to several solution approaches for our supply chain scheduling problem.

In Chapters 2 and 3 we consider the design of a production plan. In both chapters we assume that the purchase plan is given. As a consequence the purchase and the inventory costs are fixed. Therefore, the problem is to find the starting times for all operations such that all constraints are satisfied and the tardiness costs are minimized.

In Chapter 2 we also assume that each produced or purchased nonrenewable is pre-assigned to a consuming operation. We handle the resulting problem with a tabu search algorithm that generalizes the Nowicki and Smutnicki [1996] approach for the classical job shop scheduling problem by extending the classical disjunctive graph representation. The starting times are now determined by solving maximum cost flow problems in an efficient way. The solution of the maximum cost flow problem is used to define an efficient neighborhood function.

In Chapter 3 we extend the problem considered in Chapter 2 by introducing the additional problem to assign the produced or purchased nonrenewables to the consuming operations. We handle the resulting problem with a tabu search algorithm. To account the nonrenewable resource constraints another representation method is required: the activity list. Given an activity list a schedule can be generated by a list scheduling algorithm. All neighbors are modifications of the activity list.

In Chapter 4 we consider the design of a purchase plan. Hereby we assume that the production plan is given. As a consequence we know how much of each item we need and the time at which we need it. Therefore, the problem is to assign all demands to a supplier and to a moment in time in such a way that all constraints are satisfied and the sum of the purchase costs and the inventory costs is minimized. A way to solve small problem instances is by mixed integer programming: we give two such formulations. Another option is using a local search approach. Our local search approach consists of two stages. In the first stage we determine an initial solution by using the solution of the linear programming relaxation of one of the mixed integer programming formulations. In the second stage we try to improve this solution by applying local search techniques. To determine the value of the neighbors we solve generalized nonlinear knapsack problems.

In Chapter 5 we treat the entire supply chain scheduling problem. We develop four integrated solution approaches. All approaches are based on the algorithms developed in the earlier chapters. Our integrated approach to handle supply chain scheduling problems works as follows. First we decompose the problem into ap-

propriate subproblems. After that, we develop local search algorithms for these subproblems. We hereby use mathematical programming techniques to employ problem-specific knowledge. Finally, the local search algorithms are integrated into one solution approach. Thereto we must specify an order in which we make small changes to the solutions of the different subproblems. For our supply chain scheduling problem we have implemented sequential and simultaneous approaches. In a sequential approach the algorithm consists of a few consecutive runs of the algorithms for the subproblems. In a simultaneous approach we allow in each step changes to the solutions of both subproblems. Our computational experiments show that, like in many supply chain scheduling problems, solving both subproblems simultaneously gives the best results.

2

Designing a production plan

In this chapter we assume that a purchase plan is given. As a consequence the purchase and the inventory costs are fixed. Furthermore, we assume that each produced or purchased nonrenewable has been assigned to a consuming operation. Therefore, we only must assign all operations to a starting time in such a way that all constraints are met, and the tardiness costs are minimized. The resulting problem is a generalization of the classical job shop scheduling problem. The main extensions are release times, positive end-start time lags, and a general precedence graph. The total weighted tardiness is the objective function.

We attack our generalized job shop scheduling problem through a two-phase method. We use a *tabu search* algorithm to search for the order in which the operations are processed on each machine. There is no reason to delay an operation. Thus finding the optimal starting times given an execution order of the operations on each machine can be done using the standard disjunctive graph representation; see Roy and Sussmann [1964]. This method requires only the computation of a single source longest path tree in an acyclic graph. For more general scheduling problems the optimal starting times given an execution order on each machine can be found by solving a sequence of *maximum flow* problems; see Faaland and Schmitt [1987]. It is even possible to determine these starting times in polynomial time by solving a *maximum cost flow* problem; see Wennink [1995, 2000]. We show that in our case the solution to the maximum cost flow problem can be determined without loss of

much computation time in comparison with the disjunctive graph approach. This is useful, since we use the solution to the maximum cost flow problem to determine the neighborhood function for our tabu search algorithm. All sequences in our neighborhood are obtained by performing a swap of two adjacent operations. We show that only swaps that possess a certain property can improve the current solution; if no such swap is available in the neighborhood, then the current solution is optimal.

This chapter is organized as follows. In Section 2.1 we describe the model and its relation to the maximum cost flow problem. Furthermore, we describe how we determine the starting times given the execution order on each machine. In Section 2.2, we present our neighborhood function. Our tabu search method is described in Section 2.3. In Section 2.4 we give extensive computational results. Finally we make some concluding remarks.

2.1 Determining the starting times

In our model we have a set of n operations, on which an arbitrary precedence relation is defined. There is a given number of machines, each of which is available from time zero onwards and can handle at most one operation at a time. For each operation u there is a release time r_u at which it becomes available, a machine m_u that has to process it, and a positive processing time p_u. If there is a precedence constraint between two operations u and v, then operation v cannot start before the completion of operation u. Some precedence constraints stipulate a *positive end-start time lag* q_{uv}: between the completion of operation u and the start of operation v at least q_{uv} time units must elapse. The objective function is based on a subset of the operations, the so called end-operations; for an end-operation u a due date d_u and a positive weight w_u are given. The *lateness* of an end-operation is defined as its completion time minus its due date. The *tardiness* of an end-operation is defined as the maximum of its lateness and zero. The problem is to assign the operations to time intervals on the machines such that the schedule is feasible and the *total weighted tardiness* is minimized. To determine the starting times given an execution order on each machine, we use the maximum cost flow problem; this method was introduced by Wennink [1995, 2000].

2.1.1 Linear programming model

To model our generalized job shop problem, we add a dummy operation s, which precedes all operations. Its starting time and its processing time are both 0. We define for some pairs of operations (u, v) constraints of the form $S_v - S_u \geq c_{uv}$ or $S_v - S_u + T_u \geq c_{uv}$. We refer to these types of constraints as *start-start constraints* and *adjusted start-start constraints*, respectively. Adjusted start-start constraints

form a generalization of start-start constraints. We use decision variables S_u and T_u to denote the starting time and the tardiness of operation u, respectively. We refer to c_{uv} as the cost of the constraint.

With start-start constraints we can model the following restrictions on the starting times S_u. If there is a release time r_u, we define for (s, u) a start-start constraint with cost r_u. If there is a precedence relation between the operations u and v, we define for (u, v) a start-start constraint with cost p_u. If there is a positive time lag q_{uv}, then p_u is replaced with $p_u + q_{uv}$. $\mathcal{A}_1$ represents the set of all ordered pairs for which a start-start constraint has been defined. If for a pair of operations (u, v) more than one start-start constraint is defined, then only the one with the largest cost c_{uv} is taken into account.

With adjusted start-start constraints we can model the *tardiness* T_u of an end-operation u, which is defined as $\max\{S_u + p_u - d_u, 0\}$. We model the tardiness by adding the adjusted start-start constraint $S_s - S_u + T_u \geq p_u - d_u$ and the nonnegativity constraint $T_u \geq 0$. In this way the tardiness is completely determined, because $w_u > 0$. $\mathcal{E}$ denotes the set of end-operations, and $\mathcal{A}_2$ denotes the set of all ordered pairs for which such an adjusted start-start constraint has been defined given by

$$\mathcal{A}_2 = \{(u, s) | u \in \mathcal{E}\}.$$

As mentioned earlier, we assume in this stage that the relative order of each pair of operations that require the same machine is given. Orienting a pair of operations is the same as determining their relative order. $\mathcal{A}_3$ denotes the set of pairs of operations that must be oriented given by

$$\mathcal{A}_3 = \{\{u, v\} | m_u = m_v,\ (u, v) \notin \mathcal{A}_1,\ (v, u) \notin \mathcal{A}_1\}.$$

After we have oriented $\{u, v\} \in \mathcal{A}_3$, we get an ordinary start-start constraint $S_v - S_u \geq p_u$. If we have oriented all pairs of operations in $\mathcal{A}_3$, then we have an *orientation*. $\mathcal{A}_3^\gamma$ will denote the set of ordered pairs that result from orientation γ; $\phi(\gamma)$ denotes the cost of an optimal schedule corresponding to γ.

Given an orientation γ, we can find optimal starting times by solving the following linear programming model; an introduction in linear programming is given by Bazaraa et al. [1990].

$$\begin{array}{lll} \phi(\gamma) = \min & \displaystyle\sum_{u \in \mathcal{E}} w_u T_u & \\ \text{s.t.} & S_v - S_u \geq c_{uv}, & \forall (u, v) \in \mathcal{A}_1 \\ & S_s - S_u + T_u \geq p_u - d_u, & \forall u \in \mathcal{E} \\ & S_v - S_u \geq p_u, & \forall (u, v) \in \mathcal{A}_3^\gamma \\ & S_s = 0 & \\ & T_u \geq 0, & \forall u \in \mathcal{E} \end{array}$$

If we define $\mathcal{A}^\gamma = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3^\gamma$ and if $\mathcal{V}$ denotes the set of all operations including s, then the dual of this problem is

$$\begin{aligned}
\phi(\gamma) = \max \quad & \sum_{(u,v)\in\mathcal{A}_1} c_{uv}X_{uv} + \sum_{u\in\mathcal{E}} (p_u - d_u)X_{us} \quad + \sum_{(u,v)\in\mathcal{A}_3^\gamma} p_u X_{uv} \\
\text{s.t.} \quad & \sum_{(u,v)\in\mathcal{A}^\gamma} X_{uv} - \sum_{(v,z)\in\mathcal{A}^\gamma} X_{vz} = 0, \qquad \forall v \in \mathcal{V} \\
& X_{us} \le w_u, \qquad \forall u \in \mathcal{E} \\
& X_{uv} \ge 0, \qquad \forall (u,v) \in \mathcal{A}^\gamma
\end{aligned}$$

The last model is a maximum cost flow problem; see Ahuja et al. [1993]. It can be represented by the following network. Each operation in $\mathcal{V}$ is represented by a node. Every ordered pair $(u, v) \in \mathcal{A}_1$ is represented by an arc from node u to node v with infinite capacity and cost c_{uv}. For every ordered pair $(u, v) \in \mathcal{A}_1$ we check if there exists a path from u to v in $(\mathcal{V}, \mathcal{A}_1 \setminus (u, v))$ with length at least c_{uv}. If this is the case, then we remove the arc (u, v). For every end-operation $u \in \mathcal{E}$ we add an arc from node u to node s with capacity w_u and cost $p_u - d_u$. For every ordered pair $(u, v) \in \mathcal{A}_3^\gamma$ we add an arc between operations u and v with infinite capacity and cost p_u; if we want to change the order of operation u and v, then we replace arc (u, v) by arc (v, u) with infinite capacity and cost p_v. The decision variables X_{uv} of the dual problem are the arc flows.

An orientation is feasible if the graph $(\mathcal{V}, \mathcal{A}_1 \cup \mathcal{A}_3^\gamma)$ is acyclic. The problem is to find a feasible orientation γ for which $\phi(\gamma)$ is minimal. In the remainder of this chapter we use the following example to illustrate our method.

Example 2.1. Our example has nine operations and three machines. Operation 4, 7, and 9 are end-operations. In the table below we give for each operation the release time, the required machine, and the processing time. For each end-operation the due date and the weight are given.

u	1	2	3	4	5	6	7	8	9
r_u	0	0	0	0	0	0	0	1	0
m_u	1	1	2	3	1	2	3	2	3
p_u	5	2	3	2	2	4	1	3	2
d_u				12			9		7
w_u				2			1		3

The precedence relations and the positive end-start time lags are given in the following table.

arc	(1,2)	(1,3)	(2,4)	(3,4)	(5,6)	(6,7)	(8,9)
q_{uv}	3	3	0	0	3	0	1

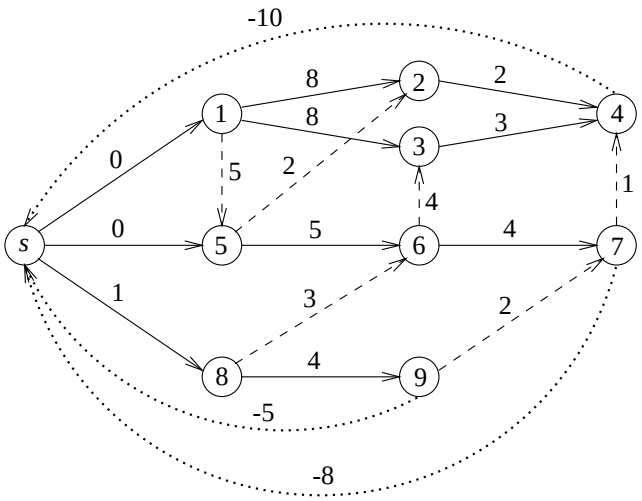

Figure 2.1: Our example graph after a orientation has been determined.

We consider the following orientation γ. The processing orders on machines 1, 2, and 3 are $1 \to 5 \to 2$, $8 \to 6 \to 3$, and $9 \to 7 \to 4$, respectively. In Figure 2.1 the resulting graph $(\mathcal{V}, \mathcal{A}^\gamma)$ is given. The solid arcs belong to $\mathcal{A}_1$, the dotted arcs belong to $\mathcal{A}_2$, and the dashed arcs belong to $\mathcal{A}_3^\gamma$. For every arc the cost is given. Notice that an oriented edge (u, v) is only in the graph if v is the immediate successor of u on m_u. □

2.1.2 Finding a maximum cost flow efficiently

Given a feasible orientation γ, we can determine optimal starting times and the corresponding objective value by solving a max cost flow problem. Since in our case $(\mathcal{V}, \mathcal{A}^\gamma)$ has a special structure we can do this without losing much computation time in comparison with the standard disjunctive graph approach. First we make the following observations about $(\mathcal{V}, \mathcal{A}^\gamma)$.

Observation 2.1. *$(\mathcal{V}, \mathcal{A}_1 \cup \mathcal{A}_3^\gamma)$ is an acyclic directed graph and all arcs of $\mathcal{A}_1 \cup \mathcal{A}_3^\gamma$ have infinite capacity and nonnegative costs.* □

Observation 2.2. *All arcs in $\mathcal{A}_2$ have finite capacity and the head of these arcs is always s.* □

Since for each node $u \in \mathcal{V}$ its inflow is equal to the outflow, the optimal flow is a union of cycles of positive costs. Observation 2.1 implies that every cycle in $(\mathcal{V}, \mathcal{A}^\gamma)$ contains at least one arc from $\mathcal{A}_2$. Observation 2.2 implies that every

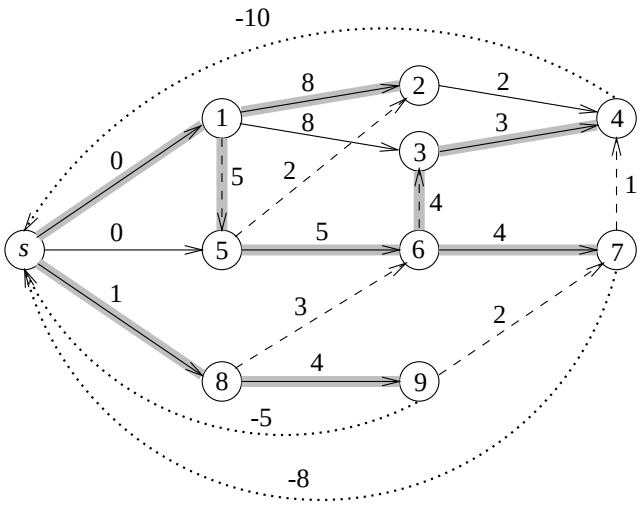

Figure 2.2: The single-source longest-path tree.

cycle contains exactly one arc from $\mathcal{A}_2$. Thus for every $(u,s) \in \mathcal{A}_2$ only the most expensive cycle that contains (u,s) is used to create the optimal flow. This cycle consists of the longest path from s to u in $(\mathcal{V}, \mathcal{A}_1 \cup \mathcal{A}_3^\gamma)$ and the arc (u,s).

This suggests the following solution method. In the graph $(\mathcal{V}, \mathcal{A}_1 \cup \mathcal{A}_3^\gamma)$ we determine a single-source longest-path tree with respect to the arc costs; this takes $O(|\mathcal{V}| + |\mathcal{A}_1 \cup \mathcal{A}_3^\gamma|)$ time. The source of this tree is sink node s. We use L_{su} to denote the longest path from s to u in the created tree. The distance from node s to node u is equal to the start time S_u of operation u. The value $\phi(\gamma)$ of the max-cost flow problem is now given by

$$\phi(\gamma) = \sum_{u \in \mathcal{E}} w_u \max\{p_u - d_u + S_u, 0\}.$$

Example 2.2. Figure 2.2 gives the single source longest path tree in our example graph. In this case $S_4 = 17$, $S_7 = 14$, and $S_9 = 5$. □

The optimal flow can now be found as follows. For every $u \in \mathcal{E}$ we check if there is a positive cycle containing (u,s). If u is tardy, i.e. $S_u + p_u - d_u > 0$, then there is a positive cycle containing (u,s) and we send w_u units of flow through L_{su} and (u,s). Otherwise we do nothing. This procedure only takes $O(|\mathcal{V}|)$ time, which is negligible in comparison with the computation time of the single-source longest path tree.

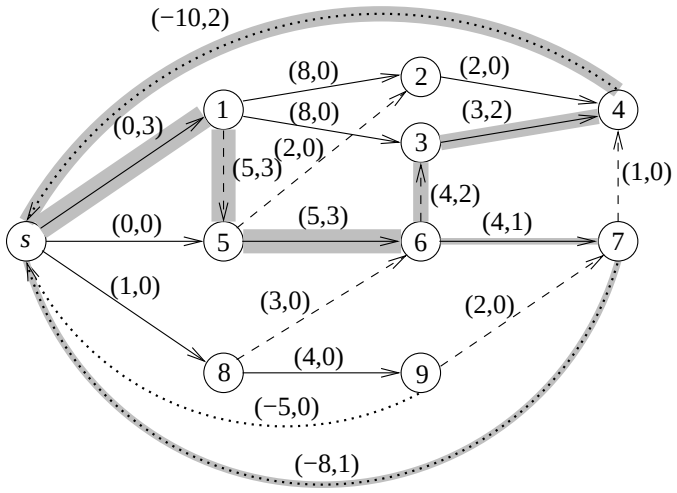

Figure 2.3: The resulting flow.

Example 2.3. Figure 2.3 gives the resulting flow for our example. For every arc the cost and the amount of flow are given in this order. The objective value is $2(17-10)+1(14-8)=20$. □

2.2 Neighborhood

In all our neighborhoods we use moves that replace an oriented edge (u,v) by (v,u). We refer to such moves as a swap of (u,v). In case of the standard job shop scheduling problem a swap of an oriented edge (u,v) results in a feasible orientation if arc (u,v) is on the longest path. Wennink [1995] generalized this lemma as follows.

Lemma 2.3. *Let γ be a feasible orientation. Swapping an oriented edge (u,v) with $X^{\gamma}_{uv} > 0$ results in a feasible orientation.* □

First we present our basic neighborhood function. Later on we present a neighborhood function that also takes into account the amount of tardiness of an end-operation and some connectivity results.

2.2.1 Basic neighborhood

Our basic neighborhood function $\mathcal{N}^{(0)}$ is a generalization of the neighborhood function for the standard job shop scheduling problem presented by Nowicki and Smutnicki [1996]. Their neighborhood function is one of the most effective neighbor-

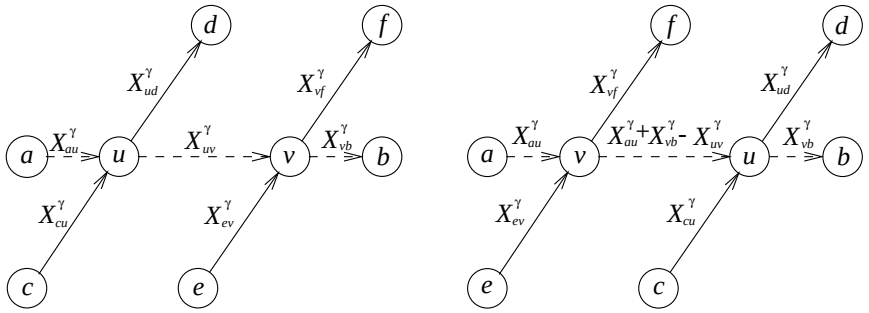

Figure 2.4: The flow before and after the transformation.

hood functions for the job shop problem. The solution to the maximum cost flow problem are used to obtain the neighbors.

For an operation u we define $\pi(u)$ as its predecessor on machine m_u and $\sigma(u)$ as its successor on machine m_u, if they exist. If u is the first operation on machine m_u, then we set $X^{\gamma}_{\pi(u)u}$ equal to $\sum_{z:(u,z)\in\mathcal{A}^{\gamma}} X^{\gamma}_{uz}$ if $S^{\gamma}_u = 0$, and 0 otherwise. Similarly, if v is the last operation on machine m_v, then we set $X^{\gamma}_{v\sigma(v)}$ equal to 0. Neighborhood function $\mathcal{N}^{(0)}$ is based on the following lemma, which states that swaps with a certain property never improve the current solution.

Lemma 2.4. *Let γ be a feasible orientation and let $(\mathcal{V}, \mathcal{A}^{\gamma})$ be the corresponding graph. Let (u, v) be an oriented edge with $X^{\gamma}_{uv} > 0$, and let γ' and $(\mathcal{V}, \mathcal{A}^{\gamma'})$ be the orientation and the graph that result after swapping (u, v). If*

$$p_v X^{\gamma}_{\pi(u)u} + p_u X^{\gamma}_{v\sigma(v)} \geq (p_v + p_u) X^{\gamma}_{uv}, \tag{2.1}$$

then $\phi(\gamma') \geq \phi(\gamma)$.

Proof. For ease of exposition, we use a and b to denote $\pi(u)$ and $\sigma(v)$, respectively. We will transform the optimal flow in $(\mathcal{V}, \mathcal{A}^{\gamma})$ into a feasible flow in $(\mathcal{V}, \mathcal{A}^{\gamma'})$ with a value greater than or equal to $\phi(\gamma)$; see Figure 2.4. With $X^{\gamma'}_{ij}$ we denote the created flow through (i, j) in $(\mathcal{V}, \mathcal{A}^{\gamma'})$. We send the following amounts of flow through the arcs (a, v), (v, u), (u, b), (a, u) and (v, b):

$$\begin{aligned} X^{\gamma'}_{av} &= X^{\gamma}_{au}, \\ X^{\gamma'}_{vu} &= X^{\gamma}_{au} + X^{\gamma}_{vb} - X^{\gamma}_{uv}, \\ X^{\gamma'}_{ub} &= X^{\gamma}_{vb}, \\ X^{\gamma'}_{au} = X^{\gamma'}_{vb} &= 0. \end{aligned}$$

All other arcs in $\mathcal{A}^{\gamma'}$ have the same amount of flow as in $\mathcal{A}^{\gamma}$. Since

$$(p_v + p_u)(X^{\gamma}_{au} + X^{\gamma}_{vb}) \geq p_v X^{\gamma}_{au} + p_u X^{\gamma}_{vb} \geq (p_v + p_u) X^{\gamma}_{uv}, \tag{2.2}$$

we have

$$X^{\gamma}_{au} + X^{\gamma}_{vb} - X^{\gamma}_{uv} \geq 0.$$

It is now easy to verify that the created flow in $(\mathcal{V}, \mathcal{A}^{\gamma'})$ is feasible. This implies that the cost of the created flow is a lower bound on the value of the optimal flow in $(\mathcal{V}, \mathcal{A}^{\gamma'})$. Using (2.2), we find that

$$\begin{gathered}\phi(\gamma') - \phi(\gamma) \geq p_v X^{\gamma'}_{vu} + p_u X^{\gamma'}_{ub} - p_u X^{\gamma}_{uv} - p_v X^{\gamma}_{vb} = \\ p_v(X^{\gamma}_{au} + X^{\gamma}_{vb} - X^{\gamma}_{uv}) + p_u X^{\gamma}_{vb} - p_u X^{\gamma}_{uv} - p_v X^{\gamma}_{vb} = \\ p_v X^{\gamma}_{au} + p_u X^{\gamma}_{vb} - (p_u + p_v) X^{\gamma}_{uv} \geq 0.\end{gathered}$$

□

We use this result to define our neighborhood $\mathcal{N}^{(0)}(\gamma)$. First we replace $X^{\gamma}_{\pi(u)u}$ and $X^{\gamma}_{v\sigma(v)}$ by 0 if $(\pi(u), u) \in \mathcal{A}_1$ and $(v, \sigma(v)) \in \mathcal{A}_1$, respectively. For every oriented edge (u, v) with $X^{\gamma}_{uv} > 0$ the following inequality

$$p_v X^{\gamma}_{\pi(u)u} + p_u X^{\gamma}_{v\sigma(v)} \geq (p_v + p_u) X^{\gamma}_{uv},$$

implies that equation 2.1 holds for the original flow. Thus Lemma 2.4 remains true after this replacement.

The neighborhood $\mathcal{N}^{(0)}(\gamma)$ consists of all orientations created by swapping an oriented edge (u, v) for which

$$p_v X^{\gamma}_{\pi(u)u} + p_u X^{\gamma}_{v\sigma(v)} < (p_v + p_u) X^{\gamma}_{uv}.$$

We now prove that $\mathcal{N}^{(0)}(\gamma) = \emptyset$ is a sufficient condition for optimality. To prove this theorem we need some preliminary lemmas.

Lemma 2.5. *If there is an oriented edge (u, v) such that $X^{\gamma}_{\pi(u)u} = 0$ and $X^{\gamma}_{uv} > 0$, then $\mathcal{N}^{(0)}(\gamma) \neq \emptyset$.*

Proof. The proof is by contradiction. Let γ' denote the orientation after swapping (u, v), and suppose that $\gamma' \notin \mathcal{N}^{(0)}(\gamma)$, that is, $p_v X^{\gamma}_{\pi(u)u} + p_u X^{\gamma}_{v\sigma(v)} \geq (p_v + p_u) X^{\gamma}_{uv}$. Since $X^{\gamma}_{\pi(u)u} = 0$ we have

$$p_u X^{\gamma}_{v\sigma(v)} \geq (p_v + p_u) X^{\gamma}_{uv}.$$

Using $X^{\gamma}_{uv} > 0$ and $p_v > 0$, we find

$$p_u X^{\gamma}_{v\sigma(v)} \geq (p_v + p_u) X^{\gamma}_{uv} > p_u X^{\gamma}_{uv},$$

and thus $X^{\gamma}_{v\sigma(v)} > X^{\gamma}_{uv}$. Since $X^{\gamma}_{uv} > 0$ this contradicts the tree-structure of the flow through $(\mathcal{V}, \mathcal{A}_1 \cup \mathcal{A}^{\gamma}_3)$. □

Lemma 2.6. *If $\mathcal{N}^{(0)}(\gamma) = \emptyset$ and u is an operation with $X^{\gamma}_{\pi(u)u} > 0$, then $X^{\gamma}_{\pi(u)u} > X^{\gamma}_{u\sigma(u)}$.*

Proof. Using the tree-structure of the optimal flow and $X^{\gamma}_{\pi(u)u} > 0$, it follows that $X^{\gamma}_{\pi(u)u} \geq X^{\gamma}_{u\sigma(u)}$. If $X^{\gamma}_{u\sigma(u)} = 0$ then the proof is trivial. Otherwise, let z denote the last operation processed on m_u such that $X^{\gamma}_{z\sigma(z)} > 0$ and $X^{\gamma}_{\pi(z)z} = X^{\gamma}_{z\sigma(z)}$; we use v to denote $\sigma(z)$. Operation v exists since $X^{\gamma}_{zv} > 0$. Now we have $X^{\gamma}_{\pi(z)z} = X^{\gamma}_{zv} > X^{\gamma}_{v\sigma(v)}$. Since the processing times are positive we have

$$p_v X^{\gamma}_{\pi(z)z} + p_z X^{\gamma}_{v\sigma(v)} < (p_z + p_v) X^{\gamma}_{zv}.$$

This implies that the orientation obtained by swapping (z, v) is in $\mathcal{N}^{(0)}(\gamma)$, which contradicts $\mathcal{N}^{(0)}(\gamma) = \emptyset$. Thus $X^{\gamma}_{\pi(u)u} > X^{\gamma}_{u\sigma(u)}$. □

Now consider L_{su}, the longest path from s to u. For every tardy end-operation e the tardiness cannot be reduced, if all arcs on L_{se} belong to $\mathcal{A}_1$. Let $\mathcal{E}^*$ denote the set of tardy end-operations e for which there is an arc on L_{se} that does not belong to $\mathcal{A}_1$. For every $e \in \mathcal{E}^*$ we denote by $z = \epsilon(e)$ the last operation on L_{se} for which $X^{\gamma}_{\pi(z)z} > 0$. The length of the longest path from $\epsilon(e)$ to e cannot be reduced since all arcs on the longest path from $\epsilon(e)$ to e also belong to $\mathcal{A}_1$.

For all operations u we define $\mathcal{E}^*_u$ as the set of $e \in \mathcal{E}^*$ for which $u = \epsilon(e)$. In Lemma 2.6 we proved that in case of an empty neighborhood $X^{\gamma}_{\pi(u)u} > X^{\gamma}_{u\sigma(u)}$ for each u with $X^{\gamma}_{\pi(u)u} > 0$. We now prove that the difference between $X^{\gamma}_{\pi(u)u}$ and $X^{\gamma}_{u\sigma(u)}$ is equal to $\sum_{e\in\mathcal{E}^*_u} w_e$.

Lemma 2.7. *If $\mathcal{N}^{(0)}(\gamma) = \emptyset$ and u is an operation with $X^{\gamma}_{\pi(u)u} > 0$, then*

$$X^{\gamma}_{\pi(u)u} - X^{\gamma}_{u\sigma(u)} = \sum_{e\in\mathcal{E}^*_u} w_e > 0.$$

Proof. As a consequence of Lemma 2.5 and the definition of $\mathcal{E}_u^*$ we can make the following observation. If operation u is on the longest path L_{se} for some $e \in \mathcal{E}^* \backslash \mathcal{E}_u^*$, then $(u, \sigma(u))$ belongs to $\mathcal{A}_3^\gamma$ and is on L_{se}. We also know that $(u, \sigma(u))$ can only be on L_{se} for some $e \in \mathcal{E}_u^*$, if $(u, \sigma(u)) \in \mathcal{A}_1$. But in that case we replaced $X^\gamma_{u\sigma(u)}$ by 0. Since the amount of flow through L_{se} is equal to w_e, we get

$$X^\gamma_{\pi(u)u} - X^\gamma_{u\sigma(u)} = \sum_{e \in \mathcal{E}_u^*} w_e.$$

As a consequence of Lemma 2.6 we have $\sum_{e \in \mathcal{E}_u^*} w_e > 0$, which completes the proof. □

We mentioned earlier that for each $e \in \mathcal{E}^*$ the length of the longest path between $\epsilon(e)$ and e cannot be reduced. Thus for every $e \in \mathcal{E}^*$ the tardiness can only be reduced if $S^\gamma_{\epsilon(e)}$ can be reduced.

We now use Smith's weighted shortest processing time rule to prove that $\sum_{e \in \mathcal{E}^*} w_e S^\gamma_{\epsilon(e)}$ cannot be reduced in case of an empty neighborhood.

Lemma 2.8. *If $\mathcal{N}^{(0)}(\gamma) = \emptyset$, then there is no orientation γ' such that*

$$\sum_{e \in \mathcal{E}^*} w_e S^{\gamma'}_{\epsilon(e)} < \sum_{e \in \mathcal{E}^*} w_e S^\gamma_{\epsilon(e)}.$$

Proof. Since for each orientation γ' we have

$$\sum_{e \in \mathcal{E}^*} w_e S^{\gamma'}_{\epsilon(e)} = \sum_{v : X^\gamma_{\pi(v)v} > 0} \sum_{e \in \mathcal{E}_v^*} w_e S^{\gamma'}_v,$$

we can prove the lemma by showing that for every machine i there is no orientation γ' such that

$$\sum_{v \in \mathcal{M}_i} \sum_{e \in \mathcal{E}_v^*} w_e S^{\gamma'}_v < \sum_{v \in \mathcal{M}_i} \sum_{e \in \mathcal{E}_v^*} w_e S^\gamma_v, \tag{2.3}$$

where $\mathcal{M}_i$ denotes $\{v | X^\gamma_{\pi(v)v} > 0 \wedge m_v = i\}$.

Let i be an arbitrary machine on which at least one operation $\epsilon(e)$ with $e \in \mathcal{E}^*$ is processed. Let z denote the last operation processed on i such that $X^\gamma_{\pi(z)z} > 0$. As a consequence of Lemma 2.5 we have $X^\gamma_{\pi(u)u} > 0$ for all predecessors u of operation z on machine i. Thus all operations on the longest path from s to z are predecessors of z on machine i, which implies that there is no idle time on machine i between time 0 and the start of operation z. Now let u be an arbitrary predecessor of operation z on machine i, and let v denote $\sigma(u)$. Since $\mathcal{N}^{(0)}(\gamma) = \emptyset$ we have

$$p_v X^\gamma_{\pi(u)u} + p_u X^\gamma_{v\sigma(v)} \geq (p_u + p_v) X^\gamma_{uv}.$$

Using Lemma (2.7) we can rewrite this as

$$\frac{\sum_{e\in\mathcal{E}_u^*} w_e}{p_u} \geq \frac{\sum_{e\in\mathcal{E}_v^*} w_e}{p_v}.$$

Thus all operations v processed on machine i with $X^{\gamma}_{\pi(v)v} > 0$ are scheduled in order of nonincreasing $\sum_{e\in\mathcal{E}_v^*} w_e/p_v$. Using the weighted shortest processing time rule of Smith [1956], we get that there is no orientation γ' such that inequality (2.3) holds. Since i is chosen arbitrarily this completes the proof. □

Now we can prove that an empty neighborhood implies optimality.

Theorem 2.9. *Let γ be a feasible orientation. If $\mathcal{N}^{(0)}(\gamma) = \emptyset$, then γ is an optimal orientation.*

Proof. Let $\mathcal{E}^T$ denote the set of tardy end-operations. We have

$$\phi(\gamma) = \sum_{e\in\mathcal{E}} w_e T_e = \sum_{e\in\mathcal{E}^*} w_e T_e + \sum_{e\in\mathcal{E}^T\setminus\mathcal{E}^*} w_e T_e + \sum_{e\in\mathcal{E}\setminus\mathcal{E}^T} w_e T_e.$$

Note that the last term of this expression is equal to zero. For every end-operation $e \in \mathcal{E}^*$ we now define $\bar{l}_{\epsilon(e)e}$ as the length of the longest path from $\epsilon(e)$ to e in $(\mathcal{V}, \mathcal{A}_1)$. Using this definition we get

$$\phi(\gamma) = \sum_{e\in\mathcal{E}^*} w_e S^{\gamma}_{\epsilon(e)} + \sum_{e\in\mathcal{E}^*} w_e(\bar{l}_{\epsilon(e)e} + p_e - d_e) + \sum_{e\in\mathcal{E}^T\setminus\mathcal{E}^*} w_e T_e.$$

For every end-operation $e \in \mathcal{E}^T \setminus \mathcal{E}^*$ its tardiness T_e cannot be reduced and for every $e \in \mathcal{E}^*$ we have that $\bar{l}_{\epsilon(e)e}$ cannot be reduced. Using Lemma 2.8 it follows that γ is an optimal orientation. □

2.2.2 More sophisticated neighborhood

Neighborhood $\mathcal{N}^{(0)}(\gamma)$ does not consider the amount of tardiness of each end-operation. If an end-operation is tardy, then it is taken into account. A way to use the amount of tardiness is to introduce a *factor* F, which is a value between 0 and 1. Only end-operations with a tardiness larger than or equal to F times the maximum tardiness $T_{\max}$ are taken into account. We employ this idea to get a more sophisticated neighborhood function $\mathcal{N}^{(F)}$ in the following way. First we define a new flow in $(\mathcal{V}, \mathcal{A}^{\gamma})$. For all $e \in \mathcal{E}$ with $p_e - d_e + S^{\gamma}_e \geq \max\{\lceil F T_{\max}\rceil, 1\}$, we

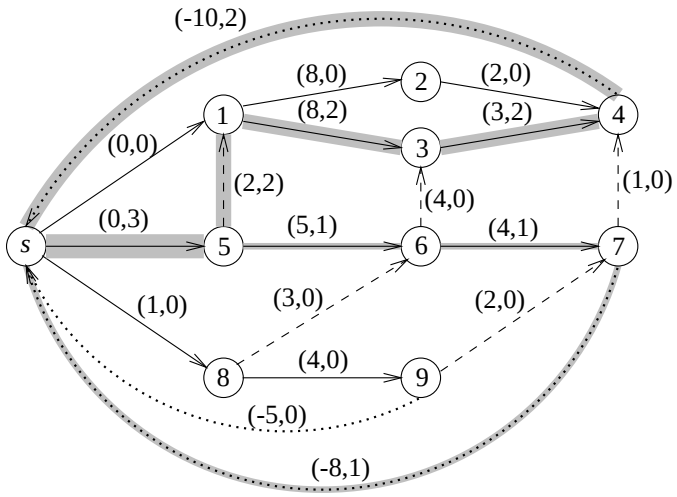

Figure 2.5: The graph after swapping arc $(1, 5)$.

send w_e units of flow through (e, s) and L_{se}, and in all other cases we do nothing. Let $X_{uv}^{\gamma(F)}$ denote the resulting flow. The neighborhood $\mathcal{N}^{(F)}(\gamma)$ consists of all orientations created by swapping the oriented edge (u, v) for which

$$p_v X_{\pi(u)u}^{\gamma(F)} + p_u X_{v\sigma(v)}^{\gamma(F)} < (p_v + p_u) X_{uv}^{\gamma(F)}.$$

If $F > F'$, then $\mathcal{N}^{(F)}(\gamma)$ does not have to be a subset of $\mathcal{N}^{(F')}(\gamma)$, but in practice $\mathcal{N}^{(F)}(\gamma)$ is usually smaller.

Example 2.4. In our example problem (see Figure 2.3) the neighborhood consists of two orientations obtained by swapping the arcs (1,5) and (6,3), respectively. In this case the factor F has no influence on the neighborhood. Now we swap $(1, 5)$. The resulting graph and flow are given in Figure 2.5. It is easy to verify that the neighborhood of the resulting orientation is empty if $F = 0$, which implies optimality. If $F > \frac{1}{3}$, then the neighborhood $\mathcal{N}^{(F)}(\gamma)$ consists of the orientation obtained by swapping arc (5,1). □

2.2.3 Connectivity of the neighborhood

On page 12 we have defined the connectivity of a neighborhood function. Our neighborhood functions have the following property.

Property 2.10. *The neighborhood function $\mathcal{N}^{(F)}$ is not optimum connected for any factor F between 0 and 1.* □

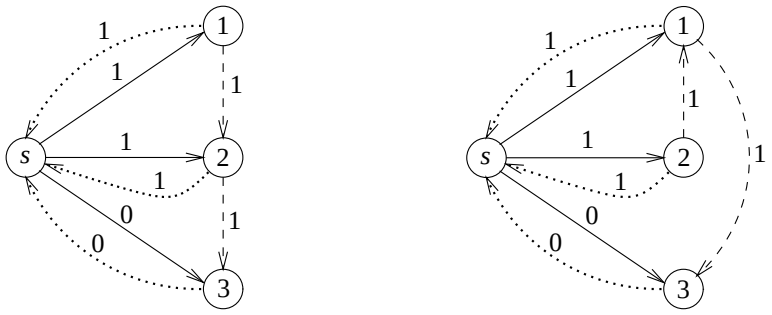

Figure 2.6: A counterexample ($\mathcal{V}$,$\mathcal{A}^{\gamma}$) and ($\mathcal{V}$,$\mathcal{A}^{\gamma'}$)).

Proof. Our counterexample has three operations and one machine. All operations are end-operations. Their release times, processing times, due dates, and weights are given below.

u	1	2	3
r_u	1	1	0
p_u	1	1	1
d_u	0	0	1
w_u	1	1	1

Let the orientation γ specify the processing order $1 \rightarrow 2 \rightarrow 3$ on machine 1. In Figure 2.6 the resulting graph $(\mathcal{V}, \mathcal{A}^{\gamma})$ is given. For every factor the neighborhood consists of the orientation obtained by swapping arc $(1, 2)$. Let γ' and $(\mathcal{V}, \mathcal{A}^{\gamma'})$ denote the resulting orientation and graph, respectively. The only neighbor of γ' is γ. But it is easy to verify that the optimal processing orders are $3 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 1 \rightarrow 2$, which cannot be reached. □

In our computational experiments we work with the neighborhood $\mathcal{N}^{(F)}(\gamma)$, where F is a factor defined beforehand. If $F > 0$ and $\mathcal{N}^{(F)}(\gamma)$ is empty, then $\mathcal{N}^{(0)}(\gamma)$ is used. In the remainder of this chapter we denote the resulting neighborhood by $\mathcal{N}(\gamma)$. All orientations in $\mathcal{N}(\gamma)$ are feasible, and if $\mathcal{N}(\gamma) = \emptyset$, then optimality has been proven. We now describe our tabu search algorithm.

2.3 Tabu search algorithm

Our algorithm starts with the construction of an initial solution. The resulting orientation γ is the input of our first iteration. In a single iteration we determine our neighborhood $\mathcal{N}(\gamma)$ and use some neighborhood searching strategy to choose a

neighbor $\gamma' \in \mathcal{N}(\gamma)$. To intensify the search we use a restarting strategy. We stop the search process when a maximum number of iterations has been reached, when all restarting possibilities are exhausted, or when optimality has been proven.

2.3.1 Finding an initial solution

To find an initial solution we use the dispatching version of the insertion algorithm of Wennink and Vaessens [Wennink and Vaessens, 1995; Wennink, 1995]. This algorithm starts with an empty schedule. In every step it randomly selects an operation u from the set of dispatchable operations. This operation is then inserted in the schedule in such a way that the increase of the maximum tardiness is as small as possible.

2.3.2 Neighborhood search strategy

In our neighborhood search strategy we allow non-improving moves to escape from local minima. We use the following procedure to choose a neighbor. Given are the current orientation γ, the nonempty neighborhood $\mathcal{N}(\gamma)$, and the current best solution ϕ^*. We evaluate the neighbors in random order and apply a *first improvement* search strategy. This means that we take the first neighbor γ' for which $\phi(\gamma') < \phi(\gamma)$. If none of the neighbors satisfies this condition, then we use some criterion to select a neighbor γ', which is created by swapping some arc (a, b). We implemented the following criteria:

Best swap (bs): select the neighbor γ' for which $\phi(\gamma')$ is minimal.

Maximum flow (fl): select the neighbor γ' for which X^{γ}_{ab} is maximal.

Combined (bf): select the neighbor γ' for which $\phi(\gamma')/X^{\gamma}_{ab}$ is minimal.

Randomized (rbf): select the neighbor γ' for which $\phi(\gamma')/X^{\gamma}_{ab}$ times a random number is minimal.

To prevent short-term cycling we use *tabu search*. In tabu search we forbid certain swaps that do not improve the best solution. When we have performed swap (a, b), then swapping all arcs of the form (b, x) is forbidden for at most N steps; N is called the *tabu tenure* and is defined dynamically to prevent long term cycling. We implemented the tabu search algorithm as follows. We introduce a mapping $\tau : \mathcal{V} \to \mathbb{N}$ and an iteration value I. Initially $\tau(u)$ is set equal to 0 for all $u \in \mathcal{V}$, and I is set equal to 0. Every iteration we increase I by at least one. Notice that the iteration value I is not the same as the number of performed iterations. A swap (a, b) is tabu if $\tau(a) > I$. Swap (a, b) is only accepted if the current best solution ϕ^* is improved or if (a, b) is not tabu. If none of the neighbors satisfies one of these criteria then we perform the swap (a, b) for which $\tau(a)$ is minimized, after which I is set equal to $\tau(a)$.

After performing the selected swap (a, b) the tabu tenure N is equal to the number of neighbors of γ plus a randomly drawn integer within the range $[0, 3]$. We finally set $\tau(b)$ equal to $I + N$ and increase I by one.

2.3.3 Restarting strategy

We have extended our tabu search algorithm with the restarting strategy presented by Nowicki and Smutnicki [1996]. Slightly modified approaches have been reported by Aarts [1996] and Verhoeven [1998]. Restarting takes place from one of the better local minima if the overall best solution has not been improved for a given number of iterations.

If we reach a local minimum that improves the overall best solution and that has at least two non-tabu neighbors, then we store that orientation γ, mapping τ, and iteration value I in a list $\mathcal{L}$. In $\mathcal{L}$ only a given number of local minima can be stored. If we want to store a new local minimum and $\mathcal{L}$ is full, then we remove the local minimum with the highest objective value.

When we restart the search process, then we select the local minimum with the lowest objective value in $\mathcal{L}$, and use its orientation, its mapping τ and its iteration value I as a restarting point for the algorithm. In the first step after the restart we select the best non-tabu neighbor, and the search process is restarted. If the number of attempted restarts from the local minimum is equal to the number of non-tabu neighbors of the local minimum minus one, then we remove the local minimum from $\mathcal{L}$.

2.4 Computational results

We tested our algorithm on two test samples. The first one is due to Pinedo and Singer [1999] and consists of 10×10 instances for the total weighted tardiness job shop scheduling problem. The second test sample was generated for the purpose of this study and consist of randomly constructed instances.

We used the following parameter settings for both samples. The maximum number of iterations is set equal to 100,000. We restart for the first time when the overall best solution has not been improved for 5,000 iterations, otherwise we restart when the overall best solution has not been improved for 2,500 iterations. The maximum size of $\mathcal{L}$ is set equal to 5. We tested our program for factor $F =$ 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 and for the four criteria mentioned in Subsection 2.3.2. We performed ten independent runs for each instance. The experiments were performed on a Sun Ultra 10, 300 MHz workstation.

	criterion *rbf*		criterion *bf*		criterion *bs*		criterion *fl*	
factor	best	average	best	average	best	average	best	average
0.0	6919	7704.9	6841	7906.9	7158	8037.3	6947	8053.1
0.2	6848	7519.1	6706	7587.7	7005	7704.0	6640	7661.2
0.4	6748	7596.0	6863	7595.7	6926	7699.2	6870	7720.7
0.6	7057	7835.4	7015	7932.1	7078	7996.5	6971	7971.8
0.8	7361	8232.5	7591	8422.5	7507	8338.8	7637	8407.9
1.0	8081	9247.2	8133	9116.7	8233	9209.3	8195	9533.1

Table 2.1: The sum of the best and the average solutions.

2.4.1 First test sample

Pinedo and Singer presented a shifting bottleneck algorithm for the total weighted tardiness job shop. They tested their algorithms on some well known 10×10 standard job shop instances to which they added due dates and weights. A 10×10 instance consists of 10 chains of 10 operations each and 10 machines. They modified the job shop instances as follows. Let $\mathcal{J}_j$ denote the set of operations that belong to job j. The due date of job j is set equal to $\lfloor 1.5 \sum_{u \in \mathcal{J}_j} p_u \rfloor$. The weights of job 1 and 2 are set equal to 4, the weights of job 3 to 8 are set equal to 2, and the weights of job 9 and 10 are set equal to 1. In our model there are no jobs, so we assigned the due dates and weights to the last operation of each chain.

To create their test sample, Pinedo and Singer modified 22 instances. ABZ5 and ABZ6 are due to Adams et al. [1988], ORB01-10 are due to Applegate and Cook [1991], FT10 is due to Fisher and Thompson [1963], and LA16-24 are due to Lawrence [1984]. LA21-24 are 15×10 instances. Therefore they eliminated the last 5 jobs to create 10×10 instances. They used branch and bound techniques to calculate the optimum for all of their instances; see Singer and Pinedo [1998].

In their shifting bottleneck approach they schedule one machine at a time. They use a branching tree to find a good order to schedule the machines. Every node of the tree represents a partial order in which the machines have been scheduled. The machine backtracking aperture defines the maximal number of branches at every node. They give test results for two variants *sb2* and *sb3*. For *sb2* and *sb3* the machine backtracking aperture is 2 and 3, respectively. Their experiments were performed on a DELL Dimension XPS P90c personal computer. Benchmark results of the Standard Performance Evaluation Corporation [1999] indicate that our workstation is 4.2 times as fast.

These instances were also used by Kreipl [2000] to test his simulated annealing approach for the total weighted tardiness job shop. He uses a neighborhood that consist of all oriented edges that are element of at least one longest path from s to a tardy end-operation. To intensify the search process he sometimes uses a smaller

						criterion *rbf*		criterion *bf*		criterion *fl*	
name	*opt*	*sb2*	*sb3*	*ls15*	*ls200*	best	average	best	average	best	average
ft10	394	394	394	485	414	394	414.7	394	428.3	394	416
abz5	69	109	77	88	70	69	78.6	70	79.4	69	73.4
abz6	0	0	0	0	0	0*	0.0	0*	0.0	0*	0.0
la16	166	178	175	166	166	166	181.3	169	204.1	166	176.9
la17	260	260	260	260	260	260	260.4	260	260.4	260	260.4
la18	34	83	34	39	34	34	34.4	34	42.8	34	40.0
la19	21	76	21	31	21	21	43.8	25	36.7	21	36.9
la20	0	0	0	1	0	0*	0.8	0*	0.9	0*	0.9
la21	0	16	0	13	0	0*	4.8	0*	4.0	0*	4.2
la22	196	196	196	196	196	196	204.1	196	203.2	196	196.0
la23	2	2	2	2	2	2	2.0	2	2.0	2	2.0
la24	82	82	82	96	90	82	89.9	88	92.4	82	96.7
orb01	1098	1539	1196	1328	1143	1231	1308.1	1124	1364.3	1141	1323.4
orb02	292	324	292	366	292	292	302.8	292	309.4	292	323.9
orb03	918	1073	967	1029	965	990	1027.9	928	1026.1	952	1035.0
orb04	358	358	358	387	358	358	446.6	358	472.6	358	642.4
orb05	405	524	517	527	455	443	491.5	478	484.6	429	513.8
orb06	426	650	426	459	426	455	528.7	426	489.9	426	490.2
orb07	50	193	50	126	119	50	63.0	50	82.0	50	78.1
orb08	1023	1298	1023	1221	1138	1023	1158.1	1023	1113.6	1035	1133.0
orb09	297	342	331	313	297	297	344.5	297	342.0	297	339.3
orb10	346	535	458	492	408	485	533.1	492	554.5	436	478.7
sum	6437	8232	6859	7625	6854	6848	7519.1	6706	7587.7	6640	7661.2

Table 2.2: Results per instance.

sb2	*sb3*	*ls15*	*ls200*	*F*=0.0	*F*=0.2	*F*=0.4	*F*=0.6	*F*=0.8	*F*=1.0
31.2	337.6	14,32	175,73	22.5	19.7	17.5	16.0	14.5	12.2

Table 2.3: Average running times.

neighborhood: in this case only oriented edges that are element of the longest path from *s* to the tardy end-operation with the highest impact on the objective value are considered. He gives test results for two variants *ls15* and *ls200*. For *ls15* and *ls200* the maximum running time is 15 and 200 seconds, respectively. His experiments were performed on a Pentium 233 MHz personal computer. Benchmark results of the Standard Performance Evaluation Corporation [1999] indicate that our workstation is 1.3 times as fast. In Table 2.1 we give the sum of the best and the average solutions for factor 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0 and the four criteria. The best results were obtained by a factor 0.2 and the *rbf*, *bf*, and *fl* criteria. In Table 2.2 we give for these algorithms the best and average results for each instance. If optimality

	criterion *rbf*		criterion *bf*		criterion *bs*		criterion *fl*	
factor	best	average	best	average	best	average	best	average
0.0	20862	23912.6	21179	24399.1	21162	24356.8	21271	24411.0
0.2	21801	25683.2	21973	25836.4	22339	25759.9	21921	25775.4
0.4	25994	30874.0	26036	30769.9	26097	30858.5	25891	31145.0
0.6	31626	37879.7	31229	37701.6	31372	37449.9	31587	38470.2
0.8	37739	47681.7	37661	47393.1	38067	47461.1	38297	46920.8
1.0	50729	68255.7	51717	68577.0	51670	68621.0	51094	68245.4

Table 2.4: The sum of the best and the average solutions.

has been proven, this is indicated by an *. Notice that this has only occurred when the optimal value is zero. Also the optimum (*opt*), the results of the shifting bottleneck algorithms of Pinedo and Singer (*sb2* and *sb3*), and the results of the simulated annealing algorithm of Kreipl (*ls15* and *ls200*) are given for each instance. One run of our algorithm is on average better than *sb2* and *ls15*, and the best of ten runs of our algorithm is slightly better than *sb3* and *ls200*.

In Table 2.3 the average running time of one run of *sb2*, *sb3*, *ls15* and *ls200* are given in real seconds, and the average running times of one run for factor 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0 are given in cpu seconds. One run of our algorithm seems to be 2.7 and 1.8 times slower than *sb2*, and *ls15*, respectively. Ten runs of our algorithm seems to be 2.5 and 1.4 times slower than *sb3* and *ls200*. Please note that our implementation was made for more general problems and does not exploit the special structure of the job shop problem for which more efficient data structures exists.

2.4.2 Second test sample

There are no instances of a generalized job shop problem like ours available in literature. To fill this gap, we have generated a class of random instances, which are publically available; see De Bontridder [2000b]. These instances were generated as follows. First we randomly generated a precedence graph using techniques described in Kolisch et al. [1995]. The processing time of each operation, the time lag of each precedence relation, and the weight of each end-operation are randomly drawn integers within $[1, 99]$, $[0, 19]$, and $[1, 4]$, respectively. Each operation is randomly assigned to a machine, and suitable release times and due dates are randomly generated. The test instances db01-10 consist each of 100 operations and 10 machines. Each instance has 10 end-operations. The instances db11-20 each have 225 operations, 15 end-operations, and 15 machines. The instances db21-30 each have 400 operations, 20 end-operations, and 20 machines.

In Table 2.4 we give the sum of the best and the average solutions for factors 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0 and the four criteria. The best results were obtained

		criterion *rbf*		criterion *bf*		criterion *bs*		criterion *fl*	
name	ub	best	average	best	average	best	average	best	average
db01	718	718	718.0	718	722.8	718	726.0	718	722.0
db02	216	216	216.0	216	216.0	216	216.0	216	216.1
db03	881	881	886.3	881	881.0	881	881.0	881	881.0
db04	104	104	104.0	104	104.0	104	104.0	104	104.0
db05	320	320	357.8	320	347.0	320	352.4	320	354.0
db06	68	68	68.0	68	68.0	68	68.0	68	68.0
db07	932	982	1011.7	982	995.8	982	994.0	978	984.6
db08	218	218	218.0	218	218.4	218	218.0	218	218.0
db09	490	490	492.2	490	490.4	490	490.4	490	491.4
db10	606	606	624.3	610	632.2	630	657.3	616	673.2
db11	136	136	148.7	136	144.4	136	143.6	136	142.7
db12	641	662	763.8	697	807.7	669	794.0	760	831.1
db13	234	244	264.8	236	261.6	244	264.9	238	251.0
db14	1044	1094	1223.0	1152	1240.9	1117	1228.7	1088	1212.0
db15	1017	1017	1211.7	1203	1373.2	1177	1303.9	1179	1303.2
db16	262	262	287.2	265	291.7	265	302.7	268	326.8
db17	276	276	307.0	276	322.2	276	293.4	276	306.4
db18	1151	1151	1233.6	1159	1242.4	1151	1236.6	1151	1251.0
db19	66	66	93.4	66	89.0	66	89.4	66	79.4
db20	1054	1118	1212.8	1108	1206.6	1160	1264.7	1069	1179.9
db21	1714	1716	2371.0	1714	2338.6	1764	2397.3	1854	2431.6
db22	8	44	239.2	44	210.4	34	212.2	8	210.6
db23	1412	1543	1685.1	1470	1686.8	1412	1655.3	1488	1719.9
db24	1066	1103	1334.9	1185	1437.6	1115	1491.9	1099	1411.5
db25	434	450	531.1	452	522.0	473	544.7	434	530.7
db26	1786	1795	1973.5	1786	2035.0	1795	2043.2	1787	2006.6
db27	178	229	280.4	178	267.7	229	253.3	214	277.3
db28	386	386	476.7	426	477.3	433	479.2	427	475.7
db29	1789	1849	2198.0	1883	2277.6	1875	2225.0	1978	2256.2
db30	1118	1118	1380.4	1136	1490.8	1144	1425.7	1142	1495.1
sum	20325	20862	23912.6	21179	24399.1	21162	24356.8	21271	24411.0

Table 2.5: Results per instance.

by the algorithms with a factor 0.0. In Table 2.5 we give per instance the best known solution (ub), and the best and average results for factor 0.0 and the four criteria.

In Table 2.6 we give for the instances with respectively 100, 225, and 400 operations the average running times of one run for factors 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0 in cpu seconds.

2.5 Conclusion

In this chapter we have presented a tabu search algorithm for a generalization of the classical job shop scheduling problem, which resembles practice more closely. All

	$F = 0.0$	$F = 0.2$	$F = 0.4$	$F = 0.6$	$F = 0.8$	$F = 1.0$
db01-10	24.9	19.5	16.5	14.7	12.3	7.9
db11-20	72.6	57.9	50.6	45.8	37.5	24.5
db21-30	167.2	136.7	115.6	99.5	83.3	51.4

Table 2.6: Average cpu running time per factor.

neighbors are obtained by swapping adjacent operations, where we have identified a necessary criterion for improvement which reduces the size of the neighborhood significantly. Our tabu search algorithm seems to be effective both with respect to time and solution quality, but for the the randomly generated instances no hard conclusions can be drawn due to a lack of strong lower bounds. This is an interesting subject for future research. Nevertheless, we conclude that also for a more practical problem like this one exploiting the structure of a specific neighborhood helps.

3

Handling nonrenewable resource constraints

In this chapter we generalize the problem described in the previous chapter by dropping the assumption that the assignment of the produced or purchased nonrenewables to the consuming operations is given. Therefore, we must assign all operations to a starting time and all nonrenewables to a consuming operation in such a way that all constraints are satisfied and the tardiness costs are minimized.

We handle the resulting problem by a tabu search algorithm. To account for the introduced nonrenewable resource constraints we use another representation method for the schedules: the activity list. Given an activity list a schedule can be generated by a list scheduling algorithm. The list scheduling algorithm renders starting times for the operations and an assignment of the produced or purchased nonrenewables to the consuming operations. All neighbors are modifications of the activity list.

This chapter is organized as follows. In Section 3.1 we describe the list scheduling heuristic that, given an activity list, renders the starting times of the operations and an assignment of the nonrenewables. In Section 3.2 we present our neighborhood function. Our tabu search method is described in Section 3.3. In Section 3.4 we give extensive computational results. Finally, we make some concluding remarks.

3.1 Using an activity list to determine the starting times

In our model we have a set of n operations, on which an arbitrary precedence relation is defined. There are a given number of machines, each of which is available from time zero onwards and can handle at most one operation at a time, and a given number of nonrenewables. Each operation u has a release time r_u at which it becomes available, a machine m_u that has to process it, and a positive processing time p_u. If there is a precedence constraint between two operations u and v, then operation v cannot start before the completion of operation u. Some precedence constraints stipulate a *positive end-start time lag* q_{uv}: between the completion of operation u and the start of operation v at least q_{uv} time units must elapse. Some operations produce or consume nonrenewables. Let $\mathcal{C}_u$ and $\mathcal{P}_u$ denote the set of nonrenewables that u consumes and produces, respectively. For every nonrenewable $i \in \mathcal{C}_u$ the consumed quantity k_u^i is given. For every nonrenewable $i \in \mathcal{P}_u$ the produced quantity l_u^i and a fixed *delivery time* f_u^i are given; if operation v consumes at least one unit of nonrenewable i produced by operation u, then f_u^i time units must elapse between the completion of operation u and the start of operation v. Operation v can also consume nonrenewables that have been purchased from external suppliers. Each receipt is characterized by a triple (i, l, r); i, l, and r denote the type of nonrenewable, the purchased quantity, and the moment of the receipt, respectively. If operation v consumes at least one unit of nonrenewable from receipt (i, l, r), then v cannot start before time r.

The objective function is computed on the basis of only a subset of the operations, the so called *end-operations*. For an end-operation u a due date d_u and a weight w_u are given. The problem is to assign the operations to time intervals on the machines such that the schedule is feasible and the *total weighted tardiness* is minimized. The tardiness of an end-operation is defined as the maximum of its lateness and zero, where the lateness of an end-operation is defined as its completion time minus its due date.

We use a list scheduling heuristic to generate a schedule. The *activity list* is the order in which the operations are scheduled. Other examples of the use of activity lists in combination with local search are given by Baar et al. [1998] and Kolisch and Hess [2000]. An activity list γ is *feasible* if it has the following characteristics.

- The activity list obeys the precedence relation.
- The operations can be scheduled in order γ without creating negative stocks.

With $u \prec_\gamma v$ ($u \succ_\gamma v$) we indicate that u is positioned before (after) v in activity list γ. Given a feasible activity list, a processing order on each machine and an

assignment of each produced or purchased nonrenewable to a consuming operation can be determined in $O(n)$. Thus every feasible activity list corresponds to a unique schedule. There is always an optimal schedule that can be represented by an activity list. It is known that finding an initial feasible solution is strongly *NP*-complete; see Carlier and Rinnooy Kan [1982]. We therefore allow schedules with negative stocks during the search process. In Subsection 3.2.5 we show how we handle negative stocks, but first we consider feasible activity lists.

3.1.1 Creating a schedule

We now show how to create a schedule given an activity list γ. We schedule the operations in the order of activity list γ, where the operations are started as early as possible. An operation v that requires some nonrenewable can only start if a sufficient amount of this nonrenewable is available. We demand that the required nonrenewables are produced by an operation u for which $u \prec_\gamma v$ or that they are purchased from an external supplier: an operation consumes the nonrenewable that becomes available for consumption as early as possible. In case of a receipt a nonrenewable becomes available at its moment of receipt. Otherwise, a nonrenewable becomes available at the completion time of the producing operation plus its delivery time. We will use the following example to illustrate our method.

Example 3.1. Our example has 9 operations, 3 machines, and 2 nonrenewables. Operations 3, 8, and 9 are end-operations. In the table below we give for each operation u the release time r_u, the required machine m_u, and the processing time p_u. For each end-operation the due date d_u and the weight w_u are given. Also the produced quantities l_u^i, the consumed quantities k_u^i, and the delivery times c_u^i of both nonrenewables $i = 1, 2$ are given.

u	1	2	3	4	5	6	7	8	9
r_u	0	0	0	4	0	0	0	0	3
m_u	1	3	3	1	3	2	2	3	1
p_u	4	1	3	3	1	2	2	1	1
l_u^1		1					1		
c_u^1		4					0		
k_u^1				1	1	1		1	
l_u^2					1	1			
c_u^2					3	0			
k_u^2							1	1	
d_u			3					8	7
w_u			2					1	3

There is a receipt of one unit nonrenewable 1 at time 1 and one unit nonrenewable 1

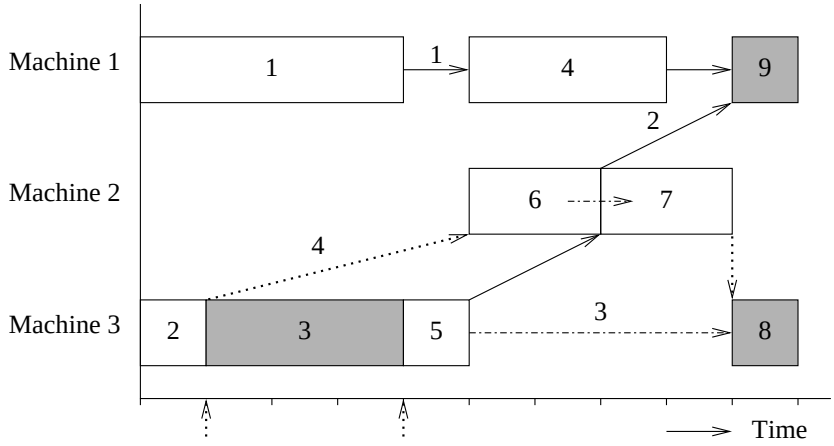

Figure 3.1: The resulting schedule for activity list 123456789.

at time 4. The precedence constraints and the positive end-start time lags are given in the following table.

arc	(1,4)	(4,9)	(5,7)	(6,9)
q_{uv}	1	0	0	2

The resulting schedule given activity list 123456789 is given in Figure 3.1. The grey operations are end-operations. The solid arcs indicate that there are precedence constraints or release times. The dotted and the dash-dot arcs indicate that there is an assignment of nonrenewable 1 or nonrenewable 2, respectively. The vertical dotted arcs under the x-axis indicate a receipt of nonrenewable 1. □

3.1.2 Left-justified schedules

It is easy to see that it is not useful to introduce unnecessary idle time. To formalize this we use again the concept of left-justified schedules; see definition 1.2. In our case there is always an optimal schedule that is left-justified. We show that there is always an optimal schedule that can be represented by an activity list by proving the following theorem.

Theorem 3.1. *Every left-justified schedule can be represented by an activity list.*

Proof. Suppose that there is a left-justified schedule that cannot be represented by an activity list. Let S_u denote the starting times in such a schedule. We create activity list γ by ordering all operations according to nondecreasing starting times. We apply our list-scheduling heuristic to activity list γ; for each operation u we denote

by S_u^γ the resulting starting time. Note that the processing order has remained the same on each machine. Since the original schedule was left-justified, there must be at least one operation u such that $S_u^\gamma > S_u$. Let z be the first such operation in γ. For all operations v for which $S_v < S_z$ we have $v \prec_\gamma z$ and $S_v^\gamma \leq S_v$, and for all operations v for which $S_v > S_z$ we have $v \succ_\gamma z$. This implies that $S_z^\gamma \leq S_z$, which is a contradiction. □

It is clear that not every activity list results in a left-justified schedule. For example, the schedule in Figure 3.1 is not left-justified: operation 6 can start at time 1 without delaying another operation.

3.2 Neighborhood

In Subsection 3.2.5 we discuss how we handle an infeasible activity list. Before that we show how we determine the neighbors of a feasible activity list. Neighbors are obtained by making one move from the current solution. We obtain a neighbor by selecting two operations a and b whose order we want to change. This is obtained by modifying the activity list appropriately. Operations a and b can be two adjacent operations on some machine, two consumers of the same type of nonrenewable, or a consumer and a producer of the same type of nonrenewable. To choose a and b we use the concept of a *critical graph*.

3.2.1 Critical graph

In the critical graph we indicate for each operation the cause of its starting time, e.g., an occupied machine, a precedence constraint, or an insufficient amount of some nonrenewable. In the critical graph we represent each operation by a node. Furthermore, we add a root node s. With $\mathcal{V}$ we denote the set of all nodes.

For each operation v we now determine the cause of its starting time. If the starting time of operation v is caused by

- a precedence constraint between u and v, then we add arc (u, v);
- a release time, then we add arc (s, v);
- the consumption of a nonrenewable i produced by u, then we add assignment arc $(u, v|i)$;
- the consumption of a nonrenewable i purchased from an external supplier, then we add assignment arc $(s, v|i)$;
- the machine predecessor u, then we add a machine arc (u, v).

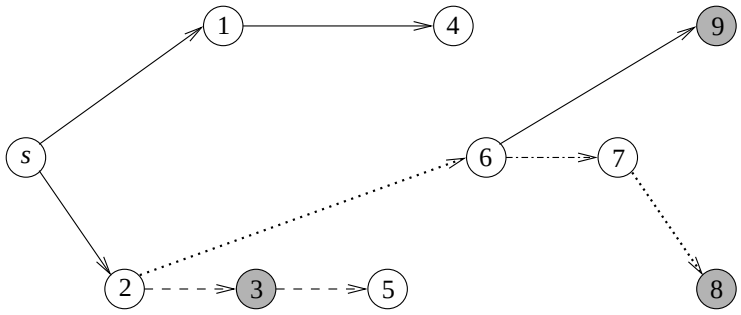

Figure 3.2: The resulting critical graph.

If the cause of the starting time is not unique, then we select the one of highest priority according to the following selection order.

1. a precedence constraint between u and v or release times;
2. the consumption of a nonrenewable i produced by machine predecessor u;
3. the machine predecessor u;
4. the consumption of a nonrenewable i.

If not all ties are settled by this selection order, then we randomly select one of the equivalent candidates. This order is required to prove some useful theorems later on.

The collection of selected arcs will be denoted by $\mathcal{A}_c$. With $\mathcal{A}_c^*$ we denote the collection of all arcs belonging to a path from s to any tardy end-operation in the critical graph $(\mathcal{V}, \mathcal{A}_c)$.

Example 3.2. In Figure 3.1 the schedule resulting from activity list 123456789 was given. The corresponding critical graph is shown in Figure 3.2. The solid arcs correspond to precedence constraints or release times. The dashed arcs correspond to machine arcs. The dotted and the dash-dot arcs are assignment arcs of nonrenewable 1 and nonrenewable 2, respectively. All end-operations are tardy. Therefore, the arcs $(s, 1)$, $(1, 4)$, and $(3, 5)$ do not belong to $\mathcal{A}_c^*$. □

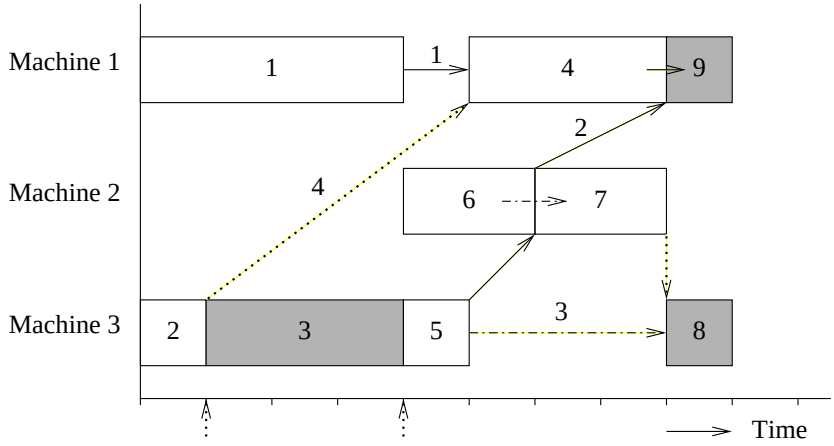

Figure 3.3: The resulting schedule for activity list 123564789.

3.2.2 Swapping two operations

As mentioned before, we obtain a neighbor by changing the order in the activity list of two selected operations a and b. Let γ be the current activity list, and let S_u^γ denote the corresponding starting times. We allow a swap of operations a and b if

- $a \prec_\gamma b$, and
- there is no precedence constraint between a and b. Notice that there is also a precedence constraint between a and b if there is a 'path' of precedence constraints between a and b.

To swap a and b we introduce a variable Q_{ab}. Depending on the type of neighbor, Q_{ab} is equal to S_b^γ or $\max\{S_a^\gamma, S_b^\gamma\}$. We modify activity list γ as follows. The order of the operations up to and including a in activity list γ remains unchanged; the same holds for the ones after b. All operations u between operation a and operation b are now evaluated in the order of the activity list γ. If $S_u^\gamma < Q_{ab}$, then u is re-inserted just before a; otherwise, u does not move. Finally we re-insert b just before a. With γ' we will denote the resulting activity list.

Example 3.3. Consider our example. Suppose that we want to swap 4 and 6. Q_{46} is in all cases equal to 5. The order of the operations 1, 2, 3, 4, 7, 8, and 9 remains unchanged in the activity list. Now we evaluate operation 5. The starting time of operation 5 is 4; therefore we insert 5 just before 4. Finally we insert 6 just before 4. Therefore, swapping operation 4 and operation 6 results in activity list 123564789. The resulting schedule is given in Figure 3.3. □

We now can state the following theorem.

Theorem 3.2. *Let γ be a feasible activity list, and let γ' be the activity list that results after swapping a and b. If there is no assignment arc between a and b and $S_a^\gamma + p_a \geq S_b^\gamma$, then activity list γ' is feasible.*

Proof. As mentioned in Section 3.1, activity list γ' is feasible if it satisfies the following criteria.

- The activity list obeys the precedence relation.
- The operations can be scheduled in order γ' without creating negative stocks.

We must prove that if $u \prec_\gamma v$ and $u \succ_{\gamma'} v$, then there is no precedence constraint between u and v and that u does not produce nonrenewables that are consumed by operation v. If there is a precedence constraint between u and v or if u produces nonrenewables that are consumed by operation v, then $S_u^\gamma + p_u \leq S_v^\gamma$. We now show that this cannot hold for any pair of operations (u, v) for which $u \prec_\gamma v$ and $u \succ_{\gamma'} v$ with the exception of (a, b). If $u \neq a$ and $v \neq b$, then

$$S_u^\gamma \geq Q_{ab} > S_v^\gamma.$$

If $u \neq a$ and $v = b$, then we have

$$S_u^\gamma + p_u \geq Q_{ab} + p_u \geq S_b^\gamma + p_u > S_b^\gamma.$$

The last case we have to consider is $u = a$ and $v \neq b$. We distinguish between the cases $Q_{ab} = \max\{S_a^\gamma, S_b^\gamma\}$ and $Q_{ab} = S_b^\gamma$. If $Q_{ab} = \max\{S_a^\gamma, S_b^\gamma\}$, then

$$S_a^\gamma + p_a \geq Q_{ab} + p_a > S_v^\gamma + p_a > S_v^\gamma.$$

If $Q_{ab} = S_b^\gamma$, then we have

$$S_a^\gamma + p_a \geq S_b^\gamma = Q_{ab} > S_v^\gamma.$$

Since there is no precedence constraint or assignment arc between a and b, this completes the proof. □

We also have identified properties that guarantee that none of the operations will be delayed in the resulting schedule after swapping two operations. These are stated in the following theorem.

Theorem 3.3. *Let γ be a feasible activity list, and let γ' be the activity list that results after swapping a and b. If $S_a^\gamma \geq S_b^\gamma$, then $S_u^{\gamma'} \leq S_u^\gamma$ for each $u \in \mathcal{V}$.*

Proof. Suppose that there is an operation v such that $S_v^{\gamma'} > S_v^\gamma$; let z be the first such operation in γ'.As a consequence of $S_a^\gamma \geq S_b^\gamma$ we have for every pair of operations (u, v) for which $u \prec_\gamma v$ and $u \succ_{\gamma'} v$ that

$$S_u^\gamma \geq Q_{ab} \geq S_v^\gamma. \tag{3.1}$$

There are only two reasons that can cause delay; we discuss them below.

- There is an operation u such that u produces nonrenewables that are consumed by operation z in schedule γ and $u \succ_{\gamma'} z$. As a consequence of the consumption we have $u \prec_\gamma z$ and $S_u^\gamma + p_u \leq S_z^\gamma$, which is in contradiction with (3.1).
- There is an operation v and a nonrenewable i such that $z \prec_\gamma v$, $z \succ_{\gamma'} v$, and nonrenewable i is consumed by both operation v and operation z. As a consequence of (3.1) we have $S_z^\gamma \geq S_v^\gamma$, which implies that the nonrenewables consumed by operation v in schedule γ are available before S_z^γ in schedule γ'. We can therefore conclude that also the second option cannot cause the delay.

□

We use these theorems to define our neighborhood function. Each neighbor is created by one of the following moves.

- The machine swap, which we discuss in Subsection 3.2.3;
- The critical assignment swap, which we will work out in Subsection 3.2.4.

3.2.3 Machine swap

For every machine arc (a, b) in $\mathcal{A}_c^*$ we create a neighbor by swapping a and b; in this case we set Q_{ab} equal to S_b^γ. Note that, since the machine arc (a, b) is in $\mathcal{A}_c^*$, we have that $a \prec_\gamma b$ and $S_a^\gamma + p_a = S_b^\gamma$. Moreover, we know that there is no precedence constraint or assignment arc between a and b, because of our selection order for choosing the critical arc of b; see Subsection 3.1.1. Hence, if activity list γ is feasible, then Theorem 3.2 shows that the swap of a and b results in a feasible activity list.

Theorem 3.4. *After performing swap (a, b), the processing order remains the same on each machine with exception of the processing order of a and b.*

Proof. As mentioned in the proof of Theorem 3.2, we have $S_u^\gamma + p_u > S_v^\gamma$ for every pair of operations (u, v) for which $u \prec_\gamma v$ and $u \succ_{\gamma'} v$ with the exception of (a, b). Therefore, u and v are not processed on the same machine, which completes the proof. □

With $\mathcal{N}_m(\gamma)$ we denote the neighbors of γ created by a machine swap.

3.2.4 Critical assignment swap

If we have two operations a and b such that $a \prec_\gamma b$ and $S_a^\gamma + p_a > S_b^\gamma$, then there is no precedence constraint or assignment arc between a and b. Thus in this case swapping a and b yields a feasible solution. After swapping two operations a and b for which $a \prec_\gamma b$ and $S_a^\gamma + p_a > S_b^\gamma$, we have $S_u^\gamma + p_u > S_v^\gamma$ for every pair of operations (u, v) for which $u \prec_\gamma v$ and $u \succ_{\gamma'} v$. As a consequence we have the following property.

Property 3.5. *After swapping two operations a and b for which $a \prec_\gamma b$ and $S_a^\gamma + p_a > S_b^\gamma$, the processing order remains the same on each machine.* □

For every operation $v \in \mathcal{V}$ for which there exists an assignment arc $(u, v|i)$ in $\mathcal{A}_c$ we create at most two neighbors.

The first type of neighbor is created by swapping v with an operation a that consumes nonrenewable i ($i \in \mathcal{C}_a$) and precedes operation v in activity list γ ($a \prec_\gamma v$). In this case Q_{av} is equal to S_v^γ. Only one such operation a is selected. In case of multiple options, we use the following order to select an operation.

1. An operation a for which $S_a^\gamma \geq S_v^\gamma$. If there is more than one candidate, then we choose the first one in the activity list. In this way all operations z for which $z \prec_\gamma v$ and $S_z^\gamma \geq S_v^\gamma$ succeeds v in activity list γ'.
2. An operation a for which $S_a^\gamma + p_a > S_v^\gamma$. In case of multiple options we choose a random candidate. In this case only the selected operation succeeds v in activity list γ'.
3. The last operation a in γ for which there is no precedence constraint between a and v. In this case swapping a and v can result in an infeasible activity list.

The second type of neighbor is created by swapping v with an operation b that produces nonrenewable i ($i \in \mathcal{P}_b$) and succeeds operation v in activity list γ ($b \succ_\gamma v$). In this case Q_{vb} is equal to $\max\{S_v^\gamma, S_b^\gamma\}$. Only one such operation b is selected. In case of multiple options, we use the following selection order to select

an operation.

1. An operation b for which $S_v^\gamma \geq S_b^\gamma$. If there is more than one candidate, then we choose the last one in the activity list. In this way all operations z for which $z \succ_\gamma v$ and $S_z^\gamma < S_v^\gamma$ precedes v in activity list γ'.
2. An operation b for which $S_v^\gamma + p_v > S_b^\gamma$. In case of multiple options we choose a random candidate. In this case only the selected operation certainly precedes v in activity list γ'.
3. The first operation b in γ for which there is no precedence constraint between v and b. In this case swapping v and b can result in an infeasible activity list.

With $\mathcal{N}_c(\gamma)$ and $\mathcal{N}_p(\gamma)$ we refer to the set of neighbors of the first and the second type, respectively. Our neighborhood $\mathcal{N}(\gamma)$ is the union of $\mathcal{N}_m(\gamma)$, $\mathcal{N}_c(\gamma)$, and $\mathcal{N}_p(\gamma)$.

Example 3.4. Consider our example. $\mathcal{N}_m(\gamma)$ consists of the neighbor obtained by machine swap $(2, 3)$. The neighbor obtained by critical assignment swap $(4, 6)$ belongs to $\mathcal{N}_c(\gamma)$. Since $S_4^\gamma \geq S_6^\gamma$ we know that none of the operations will be delayed in the resulting schedule; see Figure 3.3. This is also the reason why we selected critical assignment swap $(4, 6)$ and not critical assignment swap $(5, 6)$. Also the activity list obtained by swapping $(6, 8)$ belongs to $\mathcal{N}_c(\gamma)$. $\mathcal{N}_p(\gamma)$ consists of the activity list obtained by swapping $(6, 7)$. □

3.2.5 Handling an infeasible activity list

As mentioned earlier we allow activity lists that do not satisfy the nonrenewable resource constraints. An activity list is infeasible if there is at least one operation u that cannot be scheduled without causing a negative stock. To determine the neighborhood of an infeasible activity list we do not use the critical graph. To handle this situation we introduce a variable P with initial value 0 and an empty set of arcs $\mathcal{A}_c$. Every time we find an operation u whose demand cannot be satisfied, we increase the value of P with the missing quantity, and we add an assignment arc $(0, u|i)$ to $\mathcal{A}_c$. If $P > 0$, then the objective value is not equal to the total weighted tardiness, but P times a large given number. In this case $\mathcal{A}_c^*$ remains empty, and therefore there are no neighbors that are created through a machine swap. The neighbors obtained by critical assignment swaps are determined as described in Subsection 3.2.4.

3.3 Tabu search algorithm

Our algorithm starts with the construction of an initial solution. The resulting activity list γ is the input of our first iteration. In a single iteration we determine our neighborhood $\mathcal{N}(\gamma)$ and use a neighborhood search strategy to choose a neighbor $\gamma' \in \mathcal{N}(\gamma)$. To intensify the search we use a restarting strategy. We stop the search, when a maximum number of iterations has been reached, or when all restarting possibilities are exhausted.

3.3.1 Finding an initial solution

We randomly generate an activity list that satisfies the precedence constraints. Given this activity list we generate the initial schedule, which does not have to satisfy the nonrenewable resource constraints.

3.3.2 Neighborhood search strategy

In our neighborhood search strategy we allow non-improving moves to escape from local minima. With $\phi(\gamma)$ we will denote the objective value of activity list γ. We use the following procedure to choose a neighbor. Given are the current orientation γ, a nonempty neighborhood $\mathcal{N}(\gamma)$, and the current best solution ϕ^*. We evaluate the neighbors in random order and apply a *first improvement* search strategy. This means that we take the first neighbor γ' for which $\phi(\gamma') < \phi(\gamma)$. If none of the neighbors satisfies this condition, then we select a random neighbor γ'.

To prevent short-term cycling we use *tabu search*. In tabu search we forbid certain swaps that do not improve the best solution obtained so far. When we have performed swap (a, b), then swapping all arcs of the form (b, x), (y, a), and (a, b) is forbidden for at most N steps; N is called the *tabu tenure*.

We have implemented the tabu search algorithm as follows. We introduce the iteration count I and two mappings $\tau_1 : \mathcal{V} \to \mathbb{N}$ and $\tau_2 : \mathcal{V} \to \mathbb{N}$. Initially both $\tau_1(u)$ and $\tau_2(u)$ are set equal to 0 for all $u \in \mathcal{V}$, and I is set equal to 0. Every iteration we increase I by *at least one*. Note that the iteration count I is not the same as the number of performed iterations. A swap (a, b) is tabu if $\tau_1(a) > I$, $\tau_2(b) > I$, or if $\tau_1(b)$ is equal to $\tau_2(a)$ and $\tau_1(b) > I$. If $\tau_1(b)$ is equal to $\tau_2(a)$, then the tabu value of swap (a, b) is equal to $\max\{\tau_1(a) - I, \tau_2(b) - I, \tau_1(b) - I, 0\}$. Otherwise, the tabu value is equal to $\max\{\tau_1(a) - I, \tau_2(b) - I, 0\}$. Swap (a, b) is accepted only if the current best solution ϕ^* is improved or if (a, b) is not tabu. If none of the neighbors satisfies one of these criteria, then we perform the swap (a, b) with minimum tabu value, after which I is set equal to this tabu value.

After performing the selected swap (a, b), we set $\tau_2(a)$ and $\tau_1(b)$ equal to $I + N$ and increase I by one. In our algorithm N is equal to 8.

3.3.3 Restarting strategy

Again we have extended our tabu search algorithm with the restarting strategy presented by Nowicki and Smutnicki [1996]. Restarting takes place from one of the better local minima if the overall best solution has not been improved for a given number of iterations. If we reach a local minimum that improves the overall best solution and that has at least two non-tabu neighbors, then we store activity list γ, iteration count I, and the mappings τ_1 and τ_2 in a list $\mathcal{L}$. In $\mathcal{L}$ only a given number of local minima can be stored. If we want to store a new local minimum and $\mathcal{L}$ is full, then we remove the local minimum with the highest objective value.

When we restart the search process, we select the local minimum with the lowest objective value in $\mathcal{L}$ and use its activity list, its iteration count I, and its mappings τ_1 and τ_2 as a restarting point for the algorithm. In the first step after the restart we select the best non-tabu neighbor that has not already been attempted, and the search process is restarted. If there is only one neighbor that has not been attempted, then we remove this local minimum from $\mathcal{L}$.

3.4 Computational results

We have tested our algorithm on two test samples. The first one consists of modified job shop instances. The second test sample was randomly generated. All of the instances are available; see De Bontridder [2000a]. We used the following parameter settings for both samples. The maximum number of iterations is set equal to 100,000. We restart for the first time when the overall best solution has not been improved for 5,000 iterations; otherwise, we restart when the overall best solution has not been improved for 2,500 iterations. The maximum size of $\mathcal{L}$ is set equal to 5. We performed ten independent runs for each instance. The experiments were performed on a Sun Ultra 10 333 MHz workstation.

3.4.1 First test sample

To create the first test sample we have modified the job shop instances that were used in Singer and Pinedo [1998]; Pinedo and Singer [1999] and in Subsection 2.4.1. To these 22 instances we added nonrenewable resource constraints. For each job shop instance we generated two different instances. The instances are denoted by the name of the original instance with suffix 1 or 2. The optimum of the original instance is a lower bound on the optimum of the modified instances. To guarantee that this lower bound is not too weak we used the best available solution of the original problem (twenty of which are known to be optimal) to generate the nonrenewable resource constraints. We added nonrenewable resource constraints to the original problem in such a way that the best known solution remained feasi-

name	n	lb	ub	best	average	#best	#inf	time
abz5-1	100	69	69	69	78.3	7	0	63.3
abz6-1	100	0	0	0	0.0	10	0	43.9
ft10-1	100	394	394	394	394.0	9	1	40.7
la16-1	100	166	166	166	166.0	9	1	41.7
la17-1	100	260	260	260	260.0	9	1	34.5
la18-1	100	34	34	34	34.0	7	3	36.0
la19-1	100	21	21	21	32.4	7	1	55.5
la20-1	100	0	0	20	20.0	9	1	37.7
la21-1	100	0	0	0	0.0	9	1	45.5
la22-1	100	196	196	196	196.0	8	2	34.9
la23-1	100	2	2	2	2.0	8	2	41.9
la24-1	100	82	82	82	82.0	8	2	46.4
orb01-1	100	1098	1124	1124	1338.3	1	1	75.1
orb02-1	100	292	292	292	292.0	8	2	43.2
orb03-1	100	918	918	918	951.8	4	5	37.2
orb04-1	100	358	358	358	1386.0	3	1	46.3
orb05-1	100	405	405	405	405.0	10	0	44.0
orb06-1	100	426	426	426	476.6	4	1	57.4
orb07-1	100	50	50	50	56.8	4	2	45.7
orb08-1	100	1023	1023	1023	1182.3	3	4	34.3
orb09-1	100	297	297	297	297.0	10	0	66.0
orb10-1	100	346	424	424	424.0	8	2	60.0
Sum		6437	6541	6561	8074.5	155	33	1031.3

Table 3.1: Results per instance.

ble. In other words, the value of this solution is an upper bound for the modified instance.

In our test instances there are 10 types of nonrenewables. For each nonrenewable i the nonrenewable resource constraints were generated as follows. Each operation u consumes nonrenewable i with probability 0.1; the consumed quantity is equal to a randomly drawn integer within $U[1, 10]$. These nonrenewables are produced by an operation that is completed before the start of operation u in the used solution. The nonrenewables that are not produced by another operation are purchased from an external supplier at time 0. An operation cannot produce nonrenewable i if a direct predecessor consumes nonrenewable i. In the instances with suffix 1 operation u produces with probability 0.1 a random part of the quantity of nonrenewable i needed after the completion of operation u. In the instances with suffix 2 operation u produces with probability 0.5 the quantity of nonrenewable i needed after the completion of operation u.

As mentioned before we did 10 independent runs for each instance. In Table 3.1 and Table 3.2 we give the results for the modified job shop instances with suffix 1

name	n	lb	ub	best	average	#best	#inf	time
abz5-2	100	69	69	69	370.0	4	1	89.9
abz6-2	100	0	0	0	0.0	9	1	57.6
ft10-2	100	394	394	394	394.0	10	0	72.8
la16-2	100	166	166	166	166.0	9	1	89.7
la17-2	100	260	260	260	554.0	1	0	85.0
la18-2	100	34	34	34	42.3	8	1	58.1
la19-2	100	21	21	21	21.0	10	0	110.5
la20-2	100	0	0	20	954.1	3	2	59.0
la21-2	100	0	0	0	130.7	1	0	80.7
la22-2	100	196	196	196	325.0	5	3	63.1
la23-2	100	2	2	2	562.5	7	0	103.5
la24-2	100	82	82	82	82.0	7	3	65.8
orb01-2	100	1098	1124	1244	1549.8	1	1	122.5
orb02-2	100	292	292	292	332.0	4	3	38.6
orb03-2	100	918	918	918	4108.8	1	2	75.6
orb04-2	100	358	358	358	368.0	6	0	83.6
orb05-2	100	405	405	405	718.3	7	1	92.7
orb06-2	100	426	426	426	2403.9	1	1	86.0
orb07-2	100	50	50	50	50.0	10	0	79.3
orb08-2	100	1023	1023	1023	2760.7	1	1	106.0
orb09-2	100	297	297	297	302.5	7	2	67.8
orb10-2	100	346	424	548	1758.4	2	3	59.0
Sum		6437	6541	6805	17954.0	114	26	1746.7

Table 3.2: Results per instance.

and 2, respectively. For each instance we give the number of operations (n), the lower bound (lb), the upper bound (ub), and the best result (best). In the column 'average' the average result of the runs that resulted in a feasible solution is given; the number of runs that resulted in the best solution or in an infeasible solution are given in the sixth column (#best) and the seventh column (#inf), respectively. In the last column we give the time of one run in cpu seconds.

The results for the modified job shop instances are satisfactory. The best results of the instances with suffix 1 equal the upper bound 21 times. Also the average results are acceptable. For the instances with suffix 2 the best result equals the upper bound 19 times. The average results are less satisfactory.

3.4.2 Second test sample

We also modified the randomly generated instances used in Subsection 2.4.2. For these 30 instances we again generated two different instances with nonrenewable resource constraints, denoted by the name of the original instance with suffix 1 or 2. The nonrenewable resource constraints were generated in almost the same way as in the modified job shop instances. We again used the best available solution

name	n	ub	best	average	#best	#inf	time
db01-1	100	718	718	718.0	8	2	37.0
db02-1	100	216	216	216.0	7	3	42.6
db03-1	100	881	881	881.0	9	1	42.2
db04-1	100	104	104	104.0	8	2	36.0
db05-1	100	320	320	320.0	7	3	36.3
db06-1	100	68	68	68.0	8	2	37.1
db07-1	100	932	918	935.9	1	1	52.7
db08-1	100	218	218	245.7	1	1	49.3
db09-1	100	490	490	490.0	8	2	39.5
db10-1	100	606	606	606.8	7	2	45.8
db11-1	225	136	115	115.0	7	3	162.0
db12-1	225	641	880	1061.2	1	2	312.1
db13-1	225	234	216	247.3	1	4	156.9
db14-1	225	1044	994	1458.2	1	2	234.8
db15-1	225	1017	1427	1672.2	1	0	369.2
db16-1	225	262	270	338.2	1	2	242.0
db17-1	225	276	276	343.6	3	1	235.2
db18-1	225	1151	1151	1225.9	2	1	325.7
db19-1	225	66	66	66.0	9	1	214.8
db20-1	225	1054	1210	1365.2	1	1	373.5
db21-1	400	1714	1688	2099.8	1	0	1282.1
db22-1	400	8	10	405.4	1	0	1259.3
db23-1	400	1412	1361	1730.3	1	1	864.5
db24-1	400	1066	2117	2831.4	1	2	1601.2
db25-1	400	434	1563	1879.4	1	3	971.5
db26-1	400	1786	1795	2157.0	1	2	907.9
db27-1	400	178	225	586.0	1	2	1375.3
db28-1	400	386	494	694.0	1	5	541.5
db29-1	400	1789	1513	1747.8	1	2	1001.6
db30-1	400	1118	1569	1950.0	1	1	1301.2
Sum		20325	23479	28559.3	101	54	14150.7

Table 3.3: Results per instance.

of the original problem to generate the nonrenewable resource constraints. The only difference is that we generated delivery times within the range $[1, 20]$ and appropriate moments of receipt; these data were generated in such a way that the used solution of the original problem remained feasible.

In Table 3.3 and Table 3.4 we give the results for the randomly generated instances with suffix 1 and 2, respectively. The results for the smaller randomly generated instances are satisfactory. The best results of the instances with suffix 1 are equal to or less then the upper bound eighteen times. Seven times the upper bound was improved. The average results are acceptable too. Only the results for some of the larger instances are not satisfactory. For the small instances with suffix 2

name	n	ub	best	average	#best	#inf	time
db01-2	100	718	766	860.8	1	0	68.1
db02-2	100	216	216	271.1	2	1	83.4
db03-2	100	881	881	881.0	9	1	49.0
db04-2	100	104	104	104.8	7	0	69.4
db05-2	100	320	320	320.0	8	2	70.8
db06-2	100	68	68	68.0	10	0	90.2
db07-2	100	932	932	934.6	4	3	53.7
db08-2	100	218	218	296.3	4	1	90.7
db09-2	100	490	490	562.0	4	3	66.3
db10-2	100	606	606	667.3	3	1	67.2
db11-2	225	136	162	269.6	4	1	450.6
db12-2	225	641	1262	2246.3	1	0	589.4
db13-2	225	234	482	570.7	1	3	343.1
db14-2	225	1044	1612	2051.2	1	1	503.3
db15-2	225	1017	1912	2216.4	1	1	535.8
db16-2	225	262	705	1189.7	1	3	365.1
db17-2	225	276	474	599.6	5	1	471.0
db18-2	225	1151	1364	1590.4	1	2	446.9
db19-2	225	66	86	139.9	1	2	365.5
db20-2	225	1054	1578	1826.9	1	3	425.7
db21-2	400	1714	1924	2945.4	1	2	1440.1
db22-2	400	8	356	935.7	1	1	1831.6
db23-2	400	1412	1858	2522.1	1	0	1512.0
db24-2	400	1066	2117	2831.4	1	2	1606.8
db25-2	400	434	1563	1879.4	1	3	973.1
db26-2	400	1786	2290	2945.4	1	1	1722.6
db27-2	400	178	225	586.0	1	2	1249.1
db28-2	400	386	474	950.4	1	0	1423.8
db29-2	400	1789	1723	2573.1	1	0	2001.7
db30-2	400	1118	1670	3016.8	1	2	1773.8
Sum		20325	28438	38852.2	79	42	20739.6

Table 3.4: Results per instance.

the best result equals the upper bound eight times. The average results for these instances are satisfactory as well. For the larger instances there seems to be room for improvement.

3.5 Conclusion

In this chapter we have presented a tabu search algorithm for a generalized job shop with nonrenewable resource constraints. For instances with 100 operations the algorithm gives good results. For larger instances the algorithm is not robust and quite slow. The results could be improved by using a more sophisticated initial

solution. Furthermore, our representation method of a schedule by means of an activity-list could be used for more complicated scheduling problems; this would require a generalization of the techniques described in this chapter.

4

Designing a purchase plan

In this chapter we assume that the production plan is given. As a consequence we know how much of each item we need and the time at which we need it. Therefore, the problem is to assign all demands to a supplier and to a moment in time in such a way that all constraints are satisfied and the sum of the purchase costs and the inventory costs is minimized. In this chapter the inventory costs are the costs of holding purchased nonrenewables in inventory from the moment of receipt until the time at which we need it. We denote the resulting problem as the *purchase lot sizing* problem. Purchase lot sizing problems play an important role when suppliers specify price discounts based on the quantity ordered. In such an environment, a business can save money by grouping several orders into one large order. In order not to delay the production plan, the only option is to advance orders. However, such decisions increase the inventory of the nonrenewable. Therefore, we must find the optimal trade-off between the purchase costs and the inventory costs.

The purchase lot sizing problem is an extended version of the lot sizing problem; see Aggarwal and Park [1993] and Bahl et al. [1987]. The lot sizing problem is defined as follows. Given are a *finite time horizon*, which is divided into *periods*, and the demand for the nonrenewable in all periods. The problem is to determine for all periods the quantity (the lot size) to be produced, such that the costs of the production plan are minimized and the constraints are satisfied. In most of the articles that have appeared in the literature the costs consist of production costs and

inventory costs. In these models the production costs in a period are not affected by the produced quantities in other periods.

In our purchase lot sizing problem we consider several nonrenewables, which are delivered by external suppliers. Note that these nonrenewables only can be purchased at given moments in time. Such a moment in time marks the beginning of a new period. So a nonrenewable purchased in a period is received at the beginning of that period. We assume that the length of the periods is a given constant and that the number of periods is finite. Our goal is to find a purchase plan in such a way that the sum of the purchase costs and the inventory costs is minimized. For each nonrenewable the demand in each period and a number of alternative suppliers are given. Each supplier has its own prices, discount functions, and other restrictions. The purchased quantity of a nonrenewable in a period from a supplier must be either zero, or between a given lower and upper bound. The true cost function for buying a quantity of a nonrenewable in a time period from a supplier is piecewise linear, strictly increasing, and concave. In addition the buying firm receives discounts based on the total quantity bought from one supplier over the entire time horizon. Discounts are always based on the purchased quantity: we assume that the discount function is piecewise linear, convex, and nondecreasing. The *purchase costs* are the true costs minus the discounts. In our model the *inventory costs* are proportional to the inventory size. Notice that in our model the purchase costs of a nonrenewable in a period are affected by the purchased quantities in other periods and the purchased quantities of other nonrenewables.

Numerous articles have been published on techniques for the management and control of inventory with quantity discounts. All of these articles use enhancements of the classical economic order quantity (EOQ) concept; for an example see Rubin and Benton [1993]. In these articles a standard assumption is a constant demand rate. The problem is to find a fixed order size and a supplier for each nonrenewable such that the total costs are minimized.

In our problem we do not assume a constant demand rate; we only assume that we know the amount of each nonrenewable we need and the time at which we need it. Therefore, we must determine for each nonrenewable the quantity to be bought from each supplier in each period.

Small instances can be solved using a mixed integer linear programming approach for which we give two different formulations. Another option is using a local search approach. Our local search approach consists of two stages. In the first stage we determine an initial solution by using the solution of the linear programming relaxation of one of the mixed integer linear programming formulations. In the second stage we try to improve this solution by applying local search techniques. To determine the value of the neighbors we solve generalized nonlinear

knapsack problems.

This chapter is organized as follows. In Section 4.1 we give two mixed integer linear programming formulations for the problem. In Section 4.2 we present our neighborhood function. Our local search algorithm is described in Section 4.3. In Section 4.4 we give extensive computational results: we compare the results of our local search algorithm with the solutions of one of the mixed integer linear programming formulations. We conclude with some final remarks.

4.1 Two mixed integer linear programming formulations

The mixed integer linear programming formulations are used for two purposes. In our computational experiments we compare the results of our local search algorithm with the solutions of two mixed integer linear programming formulations. The solution of their linear programming relaxations are also used to construct initial solutions for our local search algorithm. The first formulation is rather elementary. In the second formulation we introduce a new set of decision variables, which allow us to define some strong valid inequalities. The number of variables in the second formulation is much larger than the number of variables in the first formulation.

4.1.1 Model

Let $\mathcal{T}$ denote the set of time periods and $\mathcal{I}$ the set of nonrenewables. For each nonrenewable $i \in \mathcal{I}$ we are given the demand $\overline{d}_i^t$ per time period $t \in \mathcal{T}$ and a set of suppliers $\mathcal{S}_i$.

We use two types of decision variables in our model, namely Y_{si}^t, which denotes the quantity of nonrenewable i purchased in period t from supplier s, and U_i^t, which denotes the inventory of nonrenewable i at the beginning of period t. For each quantity Y_{si}^t we are given a nonnegative lower bound $\underline{b}_{si}^t$, an upper bound $\overline{b}_{si}^t$, and a cost function α_{si}^t. The function α_{si}^t is piecewise linear, concave, and strictly increasing, and can be defined as the minimum of a number of linear functions; each linear function can be seen as one *alternative*. If $\mathcal{G}_{si}^t$ denotes the set of possible alternatives, then we have

$$\alpha_{si}^t(Y_{si}^t) = \begin{cases} \min\limits_{j \in \mathcal{G}_{si}^t} \{\widetilde{g}_{si}^{tj} Y_{si}^t + \overline{g}_{si}^{tj}\} & \text{if } Y_{si}^t > 0, \\ 0 & \text{if } Y_{si}^t = 0, \end{cases}$$

where $\widetilde{g}_{si}^{tj}$ and $\overline{g}_{si}^{tj}$ represent the unit and the fixed purchase costs for alternative j, respectively. We request that $\min_{j \in \mathcal{G}_{si}^t} \{\overline{g}_{si}^{tj}\} \geq 0$.

Each supplier s gives a discount based on a weighted sum of the purchased

quantity

$$V_s = \sum_{i:s\in\mathcal{S}_i, t\in\mathcal{T}} \widetilde{w}_{si}^t Y_{si}^t,$$

where $\widetilde{w}_{si}^t$ is the given weight for triple (s, i, t). The discount function β_s is piecewise linear, convex, and nondecreasing. A set of alternatives $\mathcal{K}_s$ is used to define the discount function given by

$$\beta_s(V_s) = \max_{a\in\mathcal{K}_s}\{\widetilde{k}_s^a V_s + \overline{k}_s^a\},$$

where $\widetilde{k}_s^a$ and $\overline{k}_s^a$ represent the unit and the fixed discounts for alternative a, respectively. We assume that $\max_{a\in\mathcal{K}_s}\{\overline{k}_s^a\} = 0$, and that $\widetilde{g}_{si}^{tj} - \widetilde{w}_{si}^t \widetilde{k}_s^a > 0$ for each possible combination. Furthermore, we assume that we can discard purchased nonrenewables without extra costs. As a consequence we have that increasing the purchased quantity of a nonrenewable never decreases the purchase costs.

In our model inventory costs are proportional to the inventory size: h_i denotes the cost of holding one unit of inventory of nonrenewable i for one time period. This results in the following model.

$$\begin{aligned}
\min \quad & \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}}\sum_{s\in\mathcal{S}_i} \alpha_{si}^t(Y_{si}^t) - \sum_{s\in\mathcal{S}} \beta_s(V_s) \quad + \quad \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}} h_i U_i^t & \\
\text{s.t.} \quad & U_i^t + \sum_{s\in\mathcal{S}_i} Y_{si}^t \geq U_i^{t+1} + \overline{d}_i^t & \forall i, t \\
& V_s = \sum_{i:s\in\mathcal{S}_i}\sum_{t\in\mathcal{T}} \widetilde{w}_{si}^t Y_{si}^t & \forall s \\
& Y_{si}^t = 0 \quad \vee \quad Y_{si}^t \geq \underline{b}_{si}^t & \forall s, i, t \\
& Y_{si}^t \leq \overline{b}_{si}^t & \forall s, i, t \\
& U_i^t \geq 0 & \forall i, t \\
& U_i^0 = 0 & \forall i
\end{aligned}$$

The first class of constraints are the traditional lot sizing constraints. There is an inequality sign, since we can discard nonrenewables without extra costs. The third type of constraint indicates that the purchased quantity must be either zero or larger than a given lower bound.

4.1.2 First formulation

We model the concave functions α_{si}^t by introducing for each alternative $j \in \mathcal{G}_{si}^t$ a binary variable Z_{si}^{tj}, which is equal to 1 if alternative j is chosen, and 0 otherwise. The variable Y_{si}^{tj} is used to indicate the amount of nonrenewable i bought from

supplier s in alternative j in period t. Alternatively, j can be chosen only if Y_{si}^{tj} belongs to the domain $[\underline{b}_{si}^{tj}, \overline{b}_{si}^{tj}]$. If $\overline{b}_{si}^{tj}$ is infinite, then we replace $\overline{b}_{si}^{tj}$ by some large integer. Now we add the following constraints.

$$\begin{aligned}
&\underline{b}_{si}^{tj} Z_{si}^{tj} \leq Y_{si}^{tj} \leq \overline{b}_{si}^{tj} Z_{si}^{tj} && \forall s, i, t, j \\
&\sum_{j \in \mathcal{G}_{si}^{t}} Z_{si}^{tj} \leq 1 && \forall s, i, t \\
&Z_{si}^{tj} \in \{0, 1\} && \forall s, i, t, j \\
&Y_{si}^{tj} \geq 0 && \forall s, i, t, j
\end{aligned}$$

The function α_{si}^{t} is now written as

$$\alpha_{si}^{t}(Y_{si}^{tj}, Z_{si}^{tj}) = \sum_{s \in \mathcal{S}_i} \sum_{j \in \mathcal{G}_{si}^{t}} (\widetilde{g}_{si}^{tj} Y_{si}^{tj} + \overline{g}_{si}^{tj} Z_{si}^{tj}).$$

To model the functions β_s we use the same techniques. Recall that the total purchased quantity is equal to

$$V_s = \sum_{i : s \in \mathcal{S}_i} \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{G}_{si}^{t}} \widetilde{w}_{si}^{t} Y_{si}^{tj}.$$

The decision variable V_s^a is equal to V_s if alternative a is chosen, and zero otherwise. The binary variable W_s^a is equal to one if $V_s^a > 0$, and zero otherwise. Now we add the following constraints.

$$\begin{aligned}
&\underline{b}_{s}^{a} W_s^a \leq V_s^a \leq \overline{b}_{s}^{a} W_s^a && \forall s, a \\
&\sum_{a \in \mathcal{K}_s} W_s^a \leq 1 && \forall s \\
&W_s^a \in \{0, 1\} && \forall s, a \\
&V_s^a \geq 0 && \forall s, a
\end{aligned}$$

The function β_s can now be written as

$$\beta_s(V_s^a, W_s^a) = \sum_{a \in \mathcal{K}_s} (\widetilde{k}_s^a V_s^a + \overline{k}_s^a W_s^a).$$

Hence, we get the following formulation.

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}}\sum_{s\in\mathcal{S}_i}\sum_{j\in\mathcal{G}_{si}^t}(\widetilde{g}_{si}^{tj}Y_{si}^{tj}+\overline{g}_{si}^{tj}Z_{si}^{tj})-\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{K}_s}(\widetilde{k}_s^a V_s^a+\overline{k}_s^a W_s^a)+\sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}}h_i U_i^t$$

$$\text{s.t.} \quad U_i^t+\sum_{s\in\mathcal{S}_i}\sum_{j\in\mathcal{G}_{si}^t}Y_{si}^{tj}\geq U_i^{t+1}+\overline{d}_i^t \qquad \forall i,t$$

$$\underline{b}_{si}^{tj}Z_{si}^{tj}\leq Y_{si}^{tj}\leq\overline{b}_{si}^{tj}Z_{si}^{tj} \qquad \forall s,i,t,j$$

$$\sum_{j\in\mathcal{G}_{si}^t}Z_{si}^{tj}\leq 1 \qquad \forall s,i,t$$

$$Z_{si}^{tj}\in\{0,1\} \qquad \forall s,i,t,j$$

$$Y_{si}^{tj}\geq 0 \qquad \forall s,i,t,j$$

$$\sum_{a\in\mathcal{K}_s}V_s^a=\sum_{i:s\in\mathcal{S}_i}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{G}_{si}^t}\widetilde{w}_{si}^t Y_{si}^{tj} \qquad \forall s$$

$$\underline{b}_s^a W_s^a\leq V_s^a\leq\overline{b}_s^a W_s^a \qquad \forall s,a$$

$$\sum_{a\in\mathcal{K}_s}W_s^a\leq 1 \qquad \forall s$$

$$W_s^a\in\{0,1\} \qquad \forall s,a$$

$$V_s^a\geq 0 \qquad \forall s,a$$

$$U_i^t\geq 0 \qquad \forall i,t$$

$$U_i^0=0 \qquad \forall i$$

We refer to this mixed integer linear programming formulation and its relaxation by MIP1 and LP1, respectively.

4.1.3 Second formulation

The first formulation is rather elementary. We now introduce a new set of decision variables, which allow us to define some strong valid inequalities. To obtain these inequalities we use techniques of Aghezzaf and Wolsey [1994]. Instead of using the variable Y_{si}^{tj} we use variables of the form $Y_{si}^{tt'j}$, which indicate the amount of nonrenewable i bought from supplier s in alternative j in period t to satisfy the demand in period t'. We get the new formulation by replacing Y_{si}^{tj} by

$$\sum_{t'\in\mathcal{T}:t'\geq t}Y_{si}^{tt'j}.$$

As a consequence of $\widetilde{g}_{si}^{tj} - \widetilde{w}_{si}^{t}\widetilde{k}_{s}^{a} > 0$, $\max\{\underline{b}_{si}^{t}, \overline{d}_{i}^{t'}\}$ is a valid upper bound for $Y_{si}^{tt'j}$. Therefore, the following constraints are valid.

$$Y_{si}^{tt'j} \leq \max\{\underline{b}_{si}^{t}, \overline{d}_{i}^{t'}\} Z_{si}^{tj} \qquad \forall s, i, t, t', j.$$

By adding these constraints we get a stronger formulation, which can be solved for small instances by a mixed integer linear programming solver.

The new variables can also be used to define the quantity of inventory of nonrenewable i. Therefore, we replace $\sum_{t\in\mathcal{T}} U_i^t$ by

$$\sum_{s\in\mathcal{S}_i}\sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}:t'\geq t}\sum_{j\in\mathcal{G}_{si}^{t}}(t'-t+1)Y_{si}^{tt'j},$$

and the first constraint by

$$\sum_{t\in\mathcal{T}:t\leq t'}\sum_{s\in\mathcal{S}_i}\sum_{j\in\mathcal{G}_{si}^{t}} Y_{si}^{tt'j} \geq \overline{d}_{i}^{t'} \qquad \forall i, t'.$$

We refer to the resulting mixed integer linear programming formulation and its relaxation by MIP2 and LP2, respectively.

4.2 Neighborhood

In our local search algorithm we consider one nonrenewable at a time. For each solution we must determine the quantity to be bought from each supplier in each period. To obtain a neighbor we shift a certain amount of demand from one period to another period or we change the assignment of the purchased quantity to the various suppliers in one period. To determine the cost of a neighbor we need to solve a nonlinear knapsack problem, which we discuss in Subsection 4.2.1. In Subsection 4.2.2 we describe the resulting neighborhood function.

4.2.1 Nonlinear knapsack problem

To determine the cost of a neighbor we must assign the purchased quantities of a nonrenewable in a period to suppliers. We determine these assignments by solving knapsack problems with piecewise linear, concave, and strictly increasing cost functions. To solve these generalized knapsack problems we use an enumerative algorithm that is based on the existence of simply structured optimal solutions. A similar approach for a generalized knapsack problem without lower bounds is given by Haberl [1999]. Our generalized knapsack problem is formally described as follows

$$
\begin{aligned}
\min \quad & \sum_{s\in\mathcal{S}} \alpha_s(Y_s) \\
\text{s.t.} \quad & \sum_{s\in\mathcal{S}} Y_s \geq d \\
& Y_s = 0 \quad \vee \quad Y_s \geq \underline{b}_s \qquad\qquad \forall s \in \mathcal{S} \\
& Y_s \leq \overline{b}_s \qquad\qquad \forall s \in \mathcal{S}
\end{aligned}
$$

For any $s \in \mathcal{S}$ the function α_s is nonnegative, piecewise linear, concave, and strictly increasing for $Y_s > 0$ with $\alpha_s(0) = 0$. By α_s^+ and α_s^- we denote the one-sided derivatives of α_s: α_s^+ and α_s^- give an indication of the additional costs of increasing Y_s by one unit and decreasing Y_s by one unit, respectively. Since α_s is concave we have $\alpha_s^+(Y_s) \geq \alpha_s^-(Y_s)$ for all $Y_s \geq 0$. For each solution δ we denote by $\mathcal{S}^\delta$ the set $\{s \mid Y_s^\delta > 0\}$. Before we describe our solution approach, we prove that there is a minimally optimal solution with a simple structure, where an optimal solution δ is minimally optimal if there does not exist an optimal solution δ' with $|\mathcal{S}^{\delta'}| < |\mathcal{S}^\delta|$.

Theorem 4.1. *There exists a minimally optimal solution δ in which there is at most one $j \in \mathcal{S}^\delta$ such that $\underline{b}_j < Y_j^\delta < \overline{b}_j$.*

Proof. Suppose to the contrary that we have for each minimally optimal solution δ at least two variables Y_i^δ and Y_j^δ such that

$$\underline{b}_i < Y_i^\delta < \overline{b}_i \quad \text{and} \quad \underline{b}_j < Y_j^\delta < \overline{b}_j.$$

Without loss of generality we may assume that $\alpha_i^-(Y_i^\delta) \geq \alpha_j^-(Y_j^\delta)$. Since α_j is a concave function we have $\alpha_i^-(Y_i^\delta) \geq \alpha_j^-(Y_j^\delta) \geq \alpha_j^+(Y_j^\delta)$. Therefore, we can increase Y_j^δ and decrease Y_i^δ by $\min\{\overline{b}_j - Y_j^\delta, Y_i^\delta - \underline{b}_i\}$ without increasing the costs. Notice that $|\mathcal{S}^\delta|$ does not change. By repeating this argument we prove the theorem. □

Theorem 4.2. *If δ is a minimally optimal solution, then for each variable Y_j^δ with $Y_j^\delta > 0$ and $\alpha_j^-(Y_j^\delta) = \max\{\alpha_s^-(Y_s^\delta) \mid s \in \mathcal{S}^\delta\}$ we have that*

$$\sum_{s\in\mathcal{S}^\delta\setminus\{j\}} \overline{b}_s < d.$$

Proof. Suppose to the contrary that there is a variable Y_j^δ for which $Y_j^\delta > 0$, $\alpha_j^-(Y_j^\delta) = \max\{\alpha_s^-(Y_s^\delta) | s \in \mathcal{S}^\delta\}$, and

$$\sum_{s \in \mathcal{S}^\delta \setminus \{j\}} \overline{b}_s \geq d.$$

Since $\alpha_j^-(Y_j^\delta) \geq \alpha_s^-(Y_s^\delta) \geq \alpha_s^+(Y_s^\delta)$ for each $s \in \mathcal{S}^\delta$, we can set Y_j^δ equal to zero and get a feasible solution with equal or lower costs by increasing Y_s^δ for some $s \in \mathcal{S}^\delta$. Since $|\mathcal{S}^\delta|$ has been decreased, we have a contradiction. □

We represent a solution by a permutation of the integer numbers 1 through $|\mathcal{S}|$; given a permutation π we generate a solution δ to our knapsack problem by the following three steps.

Step 1. For $i = 1, \ldots, |\mathcal{S}|$ we set $Y_{\pi(i)}^\delta \longleftarrow 0$.

Step 2. While $\sum_{s \in \mathcal{S}} Y_s^\delta < d$ and $j \leq |\mathcal{S}|$

$$\begin{aligned} Y_{\pi(j)}^\delta &\longleftarrow \min\{\overline{b}_{\pi(j)}, \max\{\underline{b}_{\pi(j)}, d - \sum_{s \in \mathcal{S}} Y_s^\delta\}\}; \\ j &\longleftarrow j + 1. \end{aligned}$$

Step 3. While $\sum_{s \in \mathcal{S}} Y_s^\delta > d$ and $j \geq 1$

$$\begin{aligned} Y_{\pi(j)}^\delta &\longleftarrow \max\{\underline{b}_{\pi(j)}, Y_{\pi(j)}^\delta - \sum_{s \in \mathcal{S}} Y_s^\delta + d\}; \\ j &\longleftarrow j - 1. \end{aligned}$$

Theorem 4.3. *There exists a permutation π that represents an optimal solution.*

Proof. Let δ be an optimal solution that satisfies the properties of Theorem 4.1. To prove the theorem we construct a permutation π that results in solution δ. We distinguish two cases. If there is no variable Y_i^δ with $0 < \underline{b}_i = Y_i^\delta < \overline{b}_i$, then permutation π starts with the indices s for which $Y_s^\delta = \overline{b}_s$ in an arbitrary order. These indices are followed by the index s for which $\underline{b}_s < Y_s^\delta < \overline{b}_s$. We complete permutation π with the other indices in an arbitrary order. By z we denote the last index in permutation π for which $Y_z^\delta > 0$. We have $Y_s^\delta = \overline{b}_s$ for each $s \in \mathcal{S}^\delta \setminus \{z\}$. Combined by the optimality of δ and $\alpha_z(Y_z^\delta) > 0$, we get

$$\sum_{s \in \mathcal{S}^\delta \setminus \{z\}} \overline{b}_s < d. \tag{4.1}$$

Now it is easy to see that permutation π leads to solution δ.

Otherwise, if there is a variable Y_i^δ with $0 < \underline{b}_i = Y_i^\delta < \overline{b}_i$, we start permutation π with the indices s for which $\underline{b}_s < Y_s^\delta = \overline{b}_s$ in an arbitrary order. These indices are followed by the index s for which $\underline{b}_s < Y_s^\delta < \overline{b}_s$. Thereafter, we continue with the indices of the variables for which $0 < \underline{b}_s = Y_s^\delta$ in order of nondecreasing $\alpha_s^-(Y_s^\delta)$ value. We complete permutation π with the other indices in an arbitrary order. By z we denote the last index in permutation π for which $Y_z^\delta > 0$, and by i we denote the index of an arbitrary variable for which $0 < \underline{b}_i = Y_i^\delta < \overline{b}_i$. As a consequence of the optimality of δ and $Y_i^\delta < \overline{b}_i$, we have $\alpha_s^-(Y_s^\delta) \leq \alpha_i^+(Y_i^\delta)$ for each $Y_s^\delta > \underline{b}_s$. Therefore, we have $\alpha_s^-(Y_s^\delta) \leq \alpha_i^+(Y_z^\delta) \leq \alpha_i^-(Y_i^\delta) \leq \alpha_z^-(Y_z^\delta)$ for each $Y_s^\delta > \underline{b}_s$, and thus $\alpha_z^-(Y_z^\delta) = \max\{\alpha_s^-(Y_s^\delta) | s \in \mathcal{S}^\delta\}$. By applying Theorem 4.2 we get (4.1). Now it follows that permutation π leads to solution δ. □

So we can find an optimal solution by evaluating all possible permutations of the numbers 1 through $|\mathcal{S}|$. Our generalized knapsack problem is *NP*-hard, but it can be solved by dynamic programming. From a theoretical point of view this seems much better then evaluating $|\mathcal{S}|!$ permutations, but for our instances the dynamic programming algorithm is very inefficient. Also the number of permutations evaluated by our algorithm is in practice much smaller than $|\mathcal{S}|!$.

4.2.2 Resulting neighborhood

As mentioned earlier we consider one nonrenewable at a time. Let δ' and i denote the current solution and the nonrenewable under consideration, respectively. By $D_i(t)$ we denote the quantity that in the current solution is purchased in period t to satisfy the demand restrictions. Hence, for each $t \in \mathcal{T}$ the following restriction must hold.

$$\sum_{t' \in \mathcal{T}: t' \leq t} (D_i(t') - \overline{d}_i^{t'}) \geq 0.$$

It is possible that too much of a nonrenewable is purchased in order to satisfy the lower bounds. We use $\widetilde{Y}_{si}^t$ to denote the quantity of nonrenewable i purchased in period t from supplier s that is used to satisfy the demand restrictions. Therefore, we have

$$\sum_{s \in \mathcal{S}_i} \widetilde{Y}_{si}^t = D_i(t). \tag{4.2}$$

In our local search algorithm we record the values $\widetilde{Y}_{si}^t$. Given these values we determine Y_{si}^t as follows. If $\widetilde{Y}_{si}^t = 0$, then the purchased quantity $Y_{si}^t = 0$. Otherwise the purchased quantity is calculated by

$$Y_{si}^t = \max\{\widetilde{Y}_{si}^t, \underline{b}_{si}^t\}.$$

For each pair (i,t) with $D_i(t) > 0$ we have three types of neighbors. We refer to each neighbor by $(t \Rightarrow_i t', \omega)$: each neighbor can be interpreted as moving ω units of nonrenewable i from period t to period t'. To characterize the neighbors we store the values $\Delta^-(i,t)$ and $\Delta^+(i,t')$, which we define later; the values $\Delta^-(i,t)$ and $\Delta^+(i,t')$ give an indication of the additional costs of decreasing $D_i(t)$ by one unit and increasing $D_i(t')$ by one unit, respectively. We use the values $\Delta^-(i,t)$ and $\Delta^+(i,t')$ in our local search algorithm.

The first type of neighbor is created by changing the solution in period t. In this case $D_i(t)$ remains the same for all $t \in \mathcal{T}$. The new solution for pair (i,t) is optimal given the current solution for the other pairs. If $\widehat{Y}^t_{si}$ denotes the value of Y^t_{si} in the current solution δ', then the new solution is found by solving the following problem.

$$\begin{aligned}
\min \quad & \sum_{s\in\mathcal{S}_i} \alpha^t_{si}(Y^t_{si}) - \beta_s(V_s - \widehat{Y}^t_{si} + Y^t_{si}) \\
\text{s.t.} \quad & \sum_{s\in\mathcal{S}_i} Y^t_{si} \geq D_i(t) \\
& Y^t_{si} = 0 \quad \vee \quad Y^t_{si} \geq \underline{b}^t_{si} && \forall s \in \mathcal{S}_i \\
& Y^t_{si} \leq \overline{b}^t_{si} && \forall s \in \mathcal{S}_i
\end{aligned}$$

This is a generalized knapsack problem as discussed in Subsection 4.2.1. Solving it yields an optimal set of Y^t_{si} values, but as a consequence of the lower bounds we can have an optimal solution with $\sum_{s\in\mathcal{S}_i} Y^t_{si} > D_i(t)$. We need values of $\widetilde{Y}^t_{si}$ such that equality (4.2) holds. We know that there exists an $s' \in \mathcal{S}_i$ with $Y^t_{s'i} \geq \sum_{s\in\mathcal{S}_i} Y^t_{si} - D_i(t)$. To satisfy equality (4.2) we put

$$\widetilde{Y}^t_{s'i} = Y^t_{s'i} - (\sum_{s\in\mathcal{S}_i} Y^t_{si} - D_i(t))$$

for a supplier s' for which $Y^t_{s'i} \geq \sum_{s\in\mathcal{S}_i} Y^t_{si} - D_i(t)$, and $\widetilde{Y}^t_{si} = Y^t_{si}$ for each $s \in \mathcal{S}_i \setminus \{s'\}$. We refer to this neighbor by $(t \Rightarrow_i t, 1)$. In this case $\Delta^-(i,t)$ and $\Delta^+(i,t)$ are equal to $\max\{\alpha^{t-}_{si}(\widetilde{Y}^t_{si}) - \beta^-_s(V_s) \mid \widetilde{Y}^t_{si} > 0\}$ and $\min\{\alpha^{t-}_{si}(\widetilde{Y}^t_{si}) - \beta^-_s(V_s) \mid \widetilde{Y}^t_{si} < \overline{b}^t_{si}\}$, respectively. If there is a supplier $s \in \mathcal{S}_i$ with $\widetilde{Y}^t_{si} < \underline{b}^t_{si}$, then $\Delta^+(i,t) = 0$.

The second type of neighbor is created by shifting a part of the demand to an earlier period t': a shift from t to t' is only considered if $D_i(t') < \sum_{s\in\mathcal{S}_i} \overline{b}^{t'}_{si}$ and at least one of the following two conditions holds.

- There is no t'' with $t' < t'' < t$ and $D_i(t'') > 0$.
- Period t' is the last period before t for which there exists a supplier s such that $0 < \widetilde{Y}^{t'}_{si} < \overline{b}^{t'}_{si}$.

We demand that the shifted quantity was purchased in period t from the supplier s for which $\widetilde{Y}_{si}^{t} > 0$ and $\Delta^{-}(i,t) = \alpha_{si}^{t-}(\widetilde{Y}_{si}^{t}) - \beta_{s}^{-}(V_s)$ is maximal. Therefore, $\widetilde{Y}_{si}^{t}$ is an upper bound for the shifted quantity. We refer to this supplier s by $\xi^{-}(i,t)$. A second restriction is that the shifted quantity must be purchased in period t' from one supplier. Only one supplier s' is considered. To choose the supplier we use the following selection order:

1. If there exists a supplier s with $0 < \widetilde{Y}_{si}^{t'} < \underline{b}_{si}^{t'}$, then we choose an arbitrary supplier s' with $0 < \widetilde{Y}_{s'i}^{t'} < \underline{b}_{s'i}^{t'}$. In this case $\underline{b}_{s'i}^{t'} - \widetilde{Y}_{s'i}^{t'}$ is an upper bound for the shifted quantity, and $\Delta^{+}(i,t')$ is equal to 0.
2. Otherwise if there exists a supplier s with $\underline{b}_{si}^{t'} \leq \widetilde{Y}_{si}^{t'} < \overline{b}_{si}^{t'}$, then we choose the supplier s' with $\underline{b}_{s'i}^{t'} \leq \widetilde{Y}_{s'i}^{t'} < \overline{b}_{s'i}^{t'}$ for which $\alpha_{s'i}^{t'+}(Y_{s'i}^{t'}) - \beta_{s'}^{+}(V_{s'})$ is minimal. In this case $\overline{b}_{s'i}^{t'} - \widetilde{Y}_{s'i}^{t'}$ is an upper bound for the shifted quantity, and $\Delta^{+}(i,t')$ is equal to $\alpha_{s'i}^{t'+}(Y_{s'i}^{t'}) - \beta_{s'}^{+}(V_{s'})$.
3. Otherwise, we choose the supplier s' with $0 = \widetilde{Y}_{s'i}^{t'} < \overline{b}_{s'i}^{t'}$ for which $\alpha_{s'i}^{t'+}(Y_{s'i}^{t'}) - \beta_{s'}^{+}(V_{s'})$ is minimal. In this case $\overline{b}_{s'i}^{t'} - \widetilde{Y}_{s'i}^{t'}$ is an upper bound for the shifted quantity, and $\Delta^{+}(i,t')$ is equal to $\alpha_{s'i}^{t'+}(Y_{s'i}^{t'}) - \beta_{s'}^{+}(V_{s'})$.

The shifted quantity ω is equal to the minimum of the two upper bounds. We refer to this neighbor by $(t \Rightarrow_i t', \omega)$.

The third type of neighbor is created by shifting a part of the purchased quantity to a later period t'. In this case we must take the inventories into consideration: it must be possible to perform the shift without creating negative stocks. Therefore, a shift from t to t' is only considered if $D_i(t') < \sum_{s \in \mathcal{S}_i} \overline{b}_{si}^{t'}$, $\min_{a \in \{t,\ldots,t'-1\}} \{ \sum_{j \in \mathcal{T}: j \leq a} (D_i(j) - \overline{d}_i^{j}) \} > 0$, and at least one of the following two conditions holds:

- there is no t'' such that $t < t'' < t'$ and $D_i(t'') > 0$;
- period t' is the first period after t for which $0 < \widetilde{Y}_{si}^{t'} < \overline{b}_{si}^{t'}$.

The second restriction indicates that it is possible to perform a shift without creating a negative stock if at most $\min_{a \in \{t,\ldots,t'-1\}} \{\sum_{j \in \mathcal{T}: j \leq a} (D_i(j) - \overline{d}_i^{j})\}$ is shifted from t to t'. For each neighbor the shifted quantity ω is determined in almost the same way as the shifted quantity for the second type of neighbor. The only difference is the third upper bound, which takes care of the danger of creating negative stocks. The shifted quantity ω is equal to the minimum of the three upper bounds. We refer to this neighbor by $(t \Rightarrow_i t', \omega)$.

If $t \neq t'$, then neighbor $(t \Rightarrow_i t', \omega)$ is created as follows. First we decrease

$D_i(t)$ and increase $D_i(t')$ with ω. In period t we remove this quantity from supplier $\xi^-(i,t)$. Next, we solve the nonlinear knapsack problem to determine the suppliers of the purchased quantity in period t'.

4.3 Local search algorithm

Our local search algorithm repeatedly starts independent search processes in which we consider one nonrenewable. Our local search algorithm consists of the following three steps.

Step 1. Construct an initial solution.

Step 2. Apply a search process for all of the nonrenewables in any order.

Step 3. If Step 2 has been performed a given maximum number of times, or if the best solution has not been improved in the previous run of Step 2, then stop; otherwise, go back to Step 2.

The construction of an initial solution is described in Section 4.3.1. The search process in which we consider nonrenewable i is a kind of *variable depth search*; see Kernighan and Lin [1970]. In our search strategy we start from the current best solution and generate a sequence of subsequent neighbors until some stopping criterion is satisfied. Next, a new starting point is chosen for the next sequence. In Subsection 4.3.2 we describe how we create a sequence given a starting solution. In Subsection 4.3.3 we describe how we choose the starting point for the next sequence. We stop considering nonrenewable i when a maximum number of iterations has been reached, or when all possible starting points have been used.

4.3.1 Finding an initial solution

To find an initial solution we solve one of the linear programming relaxations LP1 or LP2, which have been described in Section 4.1. In LP1 Y_{si}^{tj} indicates the amount of nonrenewable i bought from supplier s in alternative j in period t. If we use LP1, then we get as initial solution

$$\widetilde{Y}_{si}^{t} = \sum_{j \in \mathcal{G}_{si}^{t}} Y_{si}^{tj}$$

for each triple (s,i,t). The second alternative is to use the linear programming relaxation LP2. In LP2 $Y_{si}^{tt'j}$ indicates the amount of nonrenewable i bought from supplier s in alternative j in period t to satisfy the demand in period t'. In this case we get the initial solution

$$\widetilde{Y}_{si}^{t} = \sum_{t' \in \mathcal{T}: t' \geq t} \sum_{j \in \mathcal{G}_{si}^{t}} Y_{si}^{tt'j}$$

for each triple (s, i, t). For each pair (i, t) we obtain the value of $D_i(t)$ by

$$D_i(t) = \sum_{s \in \mathcal{S}_i} \widetilde{Y}_{si}^t.$$

As a consequence of $\widetilde{g}_{si}^{tj} - \widetilde{w}_{si}^t \widetilde{k}_s^a > 0$, we have for each nonrenewable i that

$$\sum_{t \in \mathcal{T}} D_i(t) = \sum_{t \in \mathcal{T}} \overline{d}_i^t.$$

Given this assignment the inventory costs are equal to

$$\sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} \sum_{t' \in \mathcal{T}: t' \leq t} h_i (D_i(t') - \overline{d}_i^{t'}).$$

The initial costs are equal to the sum of the inventory costs plus the purchase costs.

4.3.2 Generating a sequence

Given are the nonrenewable under consideration i, a start point δ, the costs of the start solution $\psi(\delta)$, a nonempty neighborhood $\mathcal{N}_i(\delta)$, and the costs of the current best solution ψ^*. In each iteration we choose a new neighbor. The costs $\psi(\delta)$ consist of two components, namely the purchase costs $\psi_p(\delta)$ and the inventory costs $\psi_i(\delta)$. If δ' is neighbor $(t \Rightarrow_i t', \omega)$, then the inventory costs are calculated by

$$\psi_i(\delta') = \psi_i(\delta) - h_i \omega (t' - t).$$

To guide the search process we use a variable I, which indicates the number of moves that have been made to obtain the current solution. By I^* we denote the value of I at the last improvement. The variables I and I^* are both set equal to zero when we start considering a new nonrenewable.

We use a *first improvement* search strategy to choose a neighbor. This means that we take the first neighbor δ' for which $\psi(\delta') < \psi(\delta)$. If none of the neighbors satisfies this criterion, then we accept the neighbor δ' for which $\psi(\delta) - \psi(\delta')$ is minimal with probability

$$e^{((I - I^*)(\psi(\delta) - \psi(\delta'))/A)}.$$

Here A is a scaling parameter. If this neighbor is not accepted, then the sequence is terminated and a new starting point is chosen.

With a first improvement search strategy the order of evaluation can affect the search process. Therefore, we evaluate the neighbors in order of nonincreasing priority: the priority of neighbor $(t \Rightarrow_i t', \omega)$ is given by $e^{((\Delta^-(i,t) - \Delta^+(i,t'))/\omega)}$ multiplied by a random number. The randomization is useful to prevent long-term

cycling, and the term $e^{((\Delta^-(i,t)-\Delta^+(i,t))/\omega)}$ is used to favor the more promising neighbors. A first improvement strategy has the positive side effect that it reduces the number of neighbors that have to be evaluated.

To prevent short-term cycling we use a kind of *tabu search*. In tabu search we temporarily forbid certain shifts: such a forbidden shift is called *tabu*. We have implemented the tabu search algorithm as follows. We introduce two mappings $\tau_1 : \mathcal{T} \to \mathbb{Z}$ and $\tau_2 : \mathcal{T} \to \mathbb{Z}$. When we consider a new nonrenewable both $\tau_1(t)$ and $\tau_2(t)$ are set equal to -1 for all $t \in \mathcal{T}$. A neighbor $(t \Rightarrow_i t', \omega)$ is tabu if

- $t' = t$, and $\tau_2(t) \geq I^* - 1$, or if
- $t' \neq t$, $\tau_1(t') = \tau_2(t)$ and $\tau_2(t) \geq I^*$.

If neighbor $(t \Rightarrow_i t', \omega)$ is accepted and $t \neq t'$, then we set $\tau_1(t)$ and $\tau_2(t')$ both equal to I. If neighbor $(t \Rightarrow_i t, \omega)$ is accepted, then we set $\tau_2(t)$ equal to I. If the best solution ψ^* is improved, then I^* is set equal to $I + 1$. Finally we increase I by one.

It is easy to prove that the size of the neighborhood $\mathcal{N}_i(\delta)$ is at most $5|\mathcal{T}|$. For instances with a large number of periods the number of evaluated neighbors and thus the computation time increases. Therefore, we have implemented a version of our local search algorithm that restricts the growth of the number of evaluated neighbors. In this version we increase the number of neighbors that are tabu. Neighbors that only change the solution in period t are considered only if $D_i(t)$ has been decreased in one of the previous steps; other neighbors $(t \Rightarrow_i t', \omega)$ are only considered if $D_i(t)$ or $D_i(t')$ has been changed in one of the previous steps. This is implemented by making neighbor $(t \Rightarrow_i t', \omega)$ also tabu if

- $I > I^*$, $t' = t$, and $\tau_1(t) < I^*$, or if
- $I > I^*$, $t' \neq t$, $\tau_1(t) < I^*$, $\tau_2(t) < I^*$, $\tau_1(t') < I^*$, and $\tau_2(t') < I^*$.

In Section 4.4 we study the effect of this restriction for instances with 52 periods.

4.3.3 Choosing a new starting point

As mentioned above, we terminate a search sequence if none of the neighbors is accepted. To choose the starting point for a new sequence we use the following method. If we reach a local minimum that improves the overall best solution and that has at least two non-tabu neighbors, then we store that solution δ, the mappings τ_1 and τ_2, and the iteration values I and I^* in a list $\mathcal{L}$.

As starting point for the new sequence we use the local minimum with the lowest objective value in $\mathcal{L}$, and use its solution δ, its mappings τ_1 and τ_2, and the iteration values I and I^* as a restarting point for the algorithm. In the first step

after the restart we select the best non-tabu neighbor that was not selected in the first step of another sequence. If all neighbors of a local minimum have been used, then we remove this local minimum from $\mathcal{L}$. The list $\mathcal{L}$ is emptied when we start considering a new nonrenewable.

4.4 Computational results

We have tested our algorithm on a randomly generated test sample, which is publically available; see De Bontridder [2001b]. We compared the results of our algorithm with the solutions of two mixed integer linear programming formulations. To solve the mixed integer linear programming programs we used CPLEX; see CPLEX [2000]. With CPLEX we also solved the linear programming relaxations which we used to obtain the initial solutions. We used the following parameter settings. Step 2 of our local search algorithm is performed at most 10 times, and we consider a nonrenewable i for at most 100 iterations. The scaling parameter A is set equal to 10. We performed 10 independent runs for each instance. The experiments were performed on a Sun Ultra 10 333 MHz workstation.

4.4.1 Test instances

Our test sample is randomly generated, where the random variables are always drawn from a uniform distribution. For each pair (i, t) the demand $\overline{d}_i^t$ is a random integer from $U[1, 100]$. Each nonrenewable i is characterized by a global price $\overline{p}_i$, which is a random integer from $U[1, 1000]$. For each nonrenewable i the unit inventory costs h_i are equal to $\lceil B\overline{p}_i \rceil$, where B is an input parameter. The probability that supplier s delivers nonrenewable i is equal to $\frac{4}{|\mathcal{S}|}$. If the number of suppliers of nonrenewable i is smaller than 3, additional suppliers are chosen. In our test instances the true cost function, the lower bound, and the upper bound are independent of the period. To generate the true and the discount cost functions we characterize each supplier s by a randomly drawn number from $U[0.9, 1.2]$. In case of a small value the prices of supplier s without incorporating any discount are low, but the amount of discount is rather small. In case of a high value supplier s has high prices and large discounts. The number of alternatives $|\mathcal{G}_{si}^t|$ and $|\mathcal{K}_s|$ are random integers from $U[1, 5]$ and $U[2, 5]$, respectively. The lower bound $\underline{b}_{si}^t$ is strictly greater than zero with probability 0.5, and the upper bound $\overline{b}_{si}^t$ is not infinite with the same probability. The fixed purchase costs are strictly greater than 0 with probability 0.5. We generated 40 instances bp01-bp40. The first 20 instances bp01-bp20 have 5 suppliers, 10 nonrenewables, and 12 periods; the other instances bp21-bp40 consist of 5 suppliers, 10 nonrenewables, and 52 periods. For the instances bp01-bp10 and bp21-bp30 the input parameter $B = 0.05$. For the other instances $B = 0.1$.

	CPLEX		algorithm LS1			algorithm LS2		
name	opt	time	best	average	time	best	average	time
bp01	2722868.7	106.3	0.03	0.08	2.7	0.00	0.00	7.6
bp02	2799213.3	356.1	0.21	0.22	2.6	0.19	0.20	8.0
bp03	3075404.8	356.1	0.02	0.05	2.6	0.02	0.04	7.2
bp04	1459908.7	74.0	0.00	0.02	3.2	0.00	0.00	6.6
bp05	2119740.7	145.2	0.24	0.24	2.5	0.20	0.20	8.6
bp06	2266288.8	263.2	0.43	0.43	2.2	0.38	0.38	7.1
bp07	2199606.1	155.7	0.07	0.10	2.6	0.00	0.00	6.2
bp08	2410382.1	99.3	0.02	0.06	2.5	0.03	0.03	5.5
bp09	2271796.1	83.1	0.02	0.18	2.0	0.08	0.08	5.9
bp10	3293095.1	89.7	0.00	0.02	2.0	0.01	0.01	6.5
average	2461830.4	172.9	0.11	0.14	2.5	0.09	0.09	6.9

Table 4.1: Results per instance.

	CPLEX		algorithm LS1			algorithm LS2		
name	opt	time	best	average	time	best	average	time
bp11	2857638.0	47.5	0.12	0.14	2.3	0.00	0.00	6.1
bp12	2857991.4	108.5	0.99	1.04	2.1	0.16	0.47	7.2
bp13	3255613.0	72.8	0.00	0.13	2.8	0.00	0.00	6.4
bp14	1486949.8	53.0	0.00	0.00	1.7	0.00	0.00	5.7
bp15	2183421.6	85.7	0.27	0.29	2.0	0.00	0.03	7.3
bp16	2346366.1	65.0	0.14	0.61	1.9	0.09	0.57	6.0
bp17	2253123.7	82.8	0.06	0.33	2.2	0.00	0.00	6.2
bp18	2455873.0	53.4	0.01	0.12	1.9	0.26	0.26	4.7
bp19	2384743.1	46.7	0.07	0.13	1.7	0.12	0.12	5.0
bp20	3528175.8	48.0	0.12	0.12	2.4	0.03	0.03	5.5
average	2560989.6	66.3	0.18	0.29	2.1	0.07	0.15	6.0

Table 4.2: Results per instance.

4.4.2 Test results

We tested the following three versions of our local search algorithm.

- LS1*:* The version that solves LP1 to obtain the initial solution.
- LS2*:* The version that solves LP2 to obtain the initial solution.
- LS3*:* The version that solves LP1 to obtain the initial solution, and reduces the number of evaluated neighbors as described at the end of Subsection 4.3.2.

In Table 4.1 and Table 4.2 we give the results for the instances with 5 suppliers, 10 nonrenewables, and 12 periods. For these instances the optimal solution was

		algorithm LS1			algorithm LS3		
name	lb	best	average	time	best	average	time
bp21	9588494.2	27.66	27.79	43.8	27.51	27.69	9.9
bp22	9659587.6	24.06	24.11	43.9	24.04	24.12	10.6
bp23	10412502.4	28.70	28.77	40.0	28.62	28.82	9.6
bp24	4872783.5	16.21	16.23	36.7	16.21	16.22	9.7
bp25	7336676.5	20.43	20.46	39.4	20.38	20.45	10.6
bp26	7977911.6	28.85	28.96	41.1	28.78	28.91	9.7
bp27	7775529.7	18.93	19.04	41.8	18.97	19.03	9.4
bp28	8672549.8	21.04	21.09	31.3	21.04	21.07	8.2
bp29	6634407.4	29.62	29.73	36.6	29.59	29.73	9.8
bp30	10686751.1	31.13	31.32	40.4	31.12	31.36	9.1
average	8361719.4	24.66	24.75	39.5	24.63	24.74	9.7

Table 4.3: Results per instance.

obtained by solving MIP2. In the second and the third column we give the optimal solution (opt) and the computation time in cpu seconds (time), respectively. We performed ten independent runs of the algorithms LS1 and LS2 on these instances, and compared the best and the average solutions of both algorithms with the optimal solutions. For both algorithms the percentage of deviation of the best solution and the average solution from the optimal value are given. In the columns 'time' for each instance the average computation time per run is given in cpu seconds. Both versions give good results in a small amount of time. The second version always needs only a few seconds. The differences between the computation times of versions LS1 and LS2 are caused by the computation of the initial solutions.

In Table 4.3 and Table 4.4 we give the results for the instances with 5 suppliers, 10 nonrenewables, and 52 periods. For these instances CPLEX was unable to find the optimal solution. These instances are so large that CPLEX even was unable to solve LP2, which renders algorithm LS2 impracticable. Therefore, we only did ten independent runs of the algorithms LS1 and LS3 on these instances. We compared the best and the average solution of both algorithms with the solution of LP1, which is a lower bound for the optimal solution. This lower bound is given in the second column. For both versions the percentage of deviation from the solution of LP1 of the best solution and the average solution are given. In the columns 'time' for each instance the average computation times per run is given in cpu seconds. It is difficult to judge the quality of the solutions. To get an indication for the quality of the lower bounds we compared the solutions of LP1 with the results of algorithm LS1 for the small instances. For bp01-bp10 and bp11-bp20 the percentage deviation is 26.61% and 31.25%, respectively. So the results seem to be good. Also notice that the quality of the solutions of algorithm LS1 and LS3 is almost the same. So,

		algorithm LS1			algorithm LS3		
name	lb	best	average	time	best	average	time
bp31	9588494.2	34.93	34.94	30.7	34.90	34.93	8.1
bp32	9659587.6	27.87	28.12	37.5	27.89	28.06	10.0
bp33	10412502.4	33.93	33.98	24.9	33.89	33.92	7.7
bp34	4872783.5	18.15	18.15	15.7	18.15	18.15	5.3
bp35	7336676.5	24.22	24.25	42.5	24.21	24.23	9.1
bp36	7977911.6	34.39	34.43	36.5	34.37	34.39	8.3
bp37	7775529.7	22.28	22.31	30.1	22.29	22.30	8.0
bp38	8672549.8	23.47	23.49	18.8	23.47	23.48	5.9
bp39	6634407.4	37.71	37.77	33.6	37.68	37.79	8.2
bp40	10686751.1	39.67	39.75	32.6	39.56	39.67	8.3
average	8361719.4	29.66	29.72	30.3	29.64	29.69	7.9

Table 4.4: Results per instance.

evaluating fewer neighbors does not affect the quality of the solution, but reduces the computation time significantly.

4.5 Conclusion

In this chapter we presented a local search algorithm for a purchase lot sizing problem with quantity discounts. For instances with 5 suppliers, 10 nonrenewables, and 12 periods we compared our approach with the solutions of a mixed integer linear programming problem solved with a standard approach. We obtained good results for instances with 5 suppliers, 10 nonrenewables, and 52 periods in a few seconds.

We made the assumption that the demand per period was given over time. In practice these demands are chosen in such a way that external deliveries do not cause delays in the production process. But there also will be some flexibility, i.e. some production orders may be moved backward without serious problems. In the next chapter we integrate the algorithms described in Chapters 3 and 4. In this way we can use the flexibility to save costs.

Our model can be extended to handle discrete order quantities. A discrete order quantity means that the order quantity should be a multiple of a given number, which can be different for each supplier. Other extensions like capacity constraints, i.e. a maximum or minimum order quantity per set of time periods, per supplier, or per time period, can also be handled by our approach.

5

Integrating purchase and production planning

In this chapter we present several approaches for the supply chain scheduling problem described in Subsection 1.2. All approaches are obtained by combining the algorithms that we have described in the previous chapters. In the supply chain scheduling problem under consideration we look at supply chains that involve a manufacturer producing items for customers. To produce these items the manufacturer has to purchase nonrenewables from external suppliers. The manufacturer must design its purchase and production plan in such a way that the total costs are minimized. These costs consist of purchase costs, inventory costs, and tardiness costs. When designing a purchase and a production plan we must take into account that the purchased nonrenewables only can be purchased at given moments in time. The formal description of the requirements for the purchase and the production plan separately, including the notation, can be found in Chapter 3 and Chapter 4, respectively. For completeness sake, we recapitulate relevant information when necessary.

This chapter is organized as follows. First, we give a straightforward approach to combine the algorithms that we have described in Chapter 3 and Chapter 4. In Section 5.2 we present some methods to improve the integration. In Section 5.3 we describe the solution approaches that we have evaluated in our computational experiments and in Section 5.4 we give extensive computational results. Finally, we make some concluding remarks.

5.1 First approach

We start with a short summary of the problem. In our model we have a set of n operations, on which an arbitrary precedence relation is defined. There are a given number of machines, each of which is available from time zero onwards and can handle at most one operation at a time, and a set of nonrenewables $\mathcal{I}$. Each operation u has a release time r_u at which it becomes available, a machine m_u that has to process it, and a positive processing time p_u. If there is a precedence constraint between two operations u and v, then operation v cannot start before the completion of operation u. Some precedence constraints stipulate a *positive end-start time lag* q_{uv}: between the completion of operation u and the start of operation v at least q_{uv} time units must elapse. Some operations produce or consume nonrenewables. Let $\mathcal{C}_u$ and $\mathcal{P}_u$ denote the set of nonrenewables that u consumes and produces, respectively. For every nonrenewable $i \in \mathcal{C}_u$ the consumed quantity k_u^i is given. For every nonrenewable $i \in \mathcal{P}_u$ the produced quantity l_u^i and the *delivery time* f_u^i are given; if some operation v consumes at least one unit of nonrenewable i produced by operation u, then f_u^i time units must elapse between the completion of operation u and the start of operation v. Operation v can also consume nonrenewables that have been purchased from external suppliers. Remark that nonrenewables can be purchased at given moments in time only. Such a moment in time marks the beginning of a new period. We assume that the length of the periods is a given constant $\widetilde{b}$ and that the number of periods is finite. The set of periods is denoted by $\mathcal{T}$. A nonrenewable purchased in period t is received at the beginning of period t, that is, at time $t\widetilde{b}$ for $t = 0, \ldots, |\mathcal{T}| - 1$.

For each nonrenewable $i \in \mathcal{I}$ we are given a set of suppliers $\mathcal{S}_i$. For the quantity of nonrenewable i purchased in period t from supplier s we are given a nonnegative lower bound $\underline{b}_{si}^t$, an upper bound $\overline{b}_{si}^t$, and a cost function α_{si}^t. The function α_{si}^t is piecewise linear, concave, and strictly increasing, and can be defined as the minimum of a number of linear functions. Each supplier s gives a discount based on the purchased quantity. The discount function β_s is piecewise linear, convex, and nondecreasing. Furthermore, we assume that we can discard purchased nonrenewables without extra costs. More details concerning the cost and discount functions are given in Chapter 3.

The tardiness costs are calculated on basis of only a subset $\mathcal{E}$ of the operations, the so-called *end-operations*. For an end-operation u a due date d_u and a weight w_u are given. The tardiness T_u of an end-operation u is defined as the maximum of its lateness and zero, where the lateness of an end-operation is defined as its completion time minus its due date. The tardiness costs are equal to $\sum_{u \in \mathcal{E}} w_u T_u$.

The inventory costs are proportional to the inventory size: h_i denotes the cost of holding one unit of inventory of nonrenewable i for one period of $\widetilde{b}$ time units.

Recall that we use the term inventory costs to denote the costs from the moment of receipt until the due date of the corresponding end-operation. The other inventory costs are included in the tardiness costs. The inventory costs are equal to a weighted sum of the due dates minus a weighted sum of the moments of receipt of the purchased nonrenewables. Note that if a nonrenewable is received after the due date of the corresponding end-operation, then a negative term is added to the inventory costs. In this way we compensate the inventory costs from the due date until the moment of receipt that are included in the tardiness costs. The due dates are given. Therefore, the inventory costs are equal to a constant minus a weighted sum of the moments of receipt of the purchased nonrenewables.

Our goal is to determine a purchase plan δ and a production plan γ in such a way that the total costs $\chi(\delta, \gamma)$ are minimized. The costs $\chi(\delta, \gamma)$ are equal to the sum of the costs of the purchase plan $\psi(\delta, \gamma)$ and the production plan $\phi(\delta, \gamma)$. The costs of the purchase plan consist of purchase and inventory costs. In this section we describe a first approach to integrate purchase and production planning, i.e., the integrated solution approach 1 (ISA1). To obtain algorithm ISA1 we combine the algorithms described in Chapters 3 and 4. The algorithm consists of the following steps.

Step 1. Design a production plan.

Step 2. Determine the demands in each period given the production plan.

Step 3. Design the purchase plan given the demands.

First, we describe the three algorithms that we use in Step 1, Step 2, and Step 3. Next, we give an overview of the complete algorithm.

5.1.1 Designing a production plan

To design a production plan a purchase plan is required. If no purchase plan δ is given, then we create an initial purchase plan by purchasing all nonrenewables that we do not produce ourselves as early as possible. To design the production plan we use the tabu search algorithm that we have described in Chapter 3. To apply the algorithm we require a list of receipts, which are described in the purchase plan. Each receipt is characterized by a triple (i, l, r), where i, l, and r denote the type of nonrenewable, the purchased quantity, and the moment of the receipt, respectively.

In our tabu search algorithm we use an activity list to represent a schedule. Given an activity list a schedule can be generated through a list scheduling algorithm. The initial solution of the algorithm can be given as input or is randomly generated in such a way that it satisfies the precedence relation. In each iteration we determine a neighborhood $\mathcal{N}(\gamma)$ and use a neighborhood search strategy to choose

a neighbor. To intensify the search we use a restarting strategy when the overall best solution has not been improved for a given number of iterations. We apply our first restart when the overall best solution has not been improved for 5,000 iterations; the next restarts occur when the overall best solution has not been improved for 2,500 iterations. We stop the search, when a maximum number of iterations $\overline{I}$ has been reached, or when all restarting possibilities are exhausted. The restarting possibilities are stored in a list $\mathcal{L}$. The number of restarts depends on the maximum size of list $\mathcal{L}$. In all our experiments the maximum size of $\mathcal{L}$ is set equal to 5. The maximum number of iterations can be extended to 100,000 if the algorithm has not obtained a feasible solution.

Running the resulting algorithm results in a solution (δ', γ'). We refer to this algorithm by

$$(\delta', \gamma') \quad \leftarrow \quad \text{MINTAR}(\delta, \gamma, \overline{I}).$$

Here, δ and γ are the initial purchase plan and activity list, respectively. If $\delta = \emptyset$, then an initial purchase plan is generated, and if $\gamma = \emptyset$, then the initial activity list is generated randomly.

5.1.2 Determining the demand

Let (δ, γ) be the current solution. Before we can design a new purchase plan we first determine for each nonrenewable i the demand $\overline{D}_i(t)$ per time period $t \in \mathcal{T}$. To obtain a good purchase plan the demands must occur as late as possible. Therefore, we calculate the demands as follows. First, we determine for each operation v the latest possible starting time $\overline{S}_v^{\delta\gamma}$ in such a way that

- the tardiness of none of the end-operations increases,
- the processing order on each machine remains the same, and
- the assignment of the produced nonrenewables to the consuming operations remains the same as in solution (δ, γ).

The purchased nonrenewables that are consumed by operation v must be received at or before time $\overline{S}_v^{\delta\gamma}$. If $\overline{S}_v^{\delta\gamma}$ belongs to period t, we add these nonrenewables to the demands at the beginning of period t. If the last period finishes before time $\overline{S}_v^{\delta\gamma}$, the required purchased nonrenewables are added to the demands at the beginning of the last period. Furthermore, we modify activity list γ by ordering all operations according to nondecreasing $\overline{S}_v^{\delta\gamma}$. It is easy to see that the resulting activity list is feasible.

Running the algorithm results in a modified activity list γ' and a list of demands $\overline{D}$. We refer to this algorithm by

$$(\gamma', \overline{D}) \quad \leftarrow \quad \text{DEMAND}(\delta, \gamma).$$

5.1.3 Designing a purchase plan

Given the demands $\overline{D}$ we can design a purchase plan by running one of the local search algorithms described in Chapter 4. There are two ways to create an initial solution for our local search algorithm. The first way is to use the solution of the linear programming relaxation LP1. The second way is to use a purchase plan δ that has been given as input. If purchase plan δ does not satisfy the demands, then we advance the receipts of some nonrenewables as little as possible such that a feasible purchase plan is obtained. In all periods where the total ordered demand has changed for any item i, we determine a new assignment of the demands to the suppliers by solving generalized knapsack problems as described in Chapter 4.

Next we try to improve the initial solution by applying local search techniques. To determine the value of the neighbors we solve generalized nonlinear knapsack problems. We use the neighborhood searching strategy that reduces the number of evaluated neighbors as described at the end of Subsection 4.3.2. The local search algorithm consists of the following steps.

Step 1. Construct an initial solution.

Step 2. Apply a search for all of the nonrenewables in any order.

Step 3. If Step 2 has been performed a given number of times H, or when the best solution has not been improved in the previous run of Step 2, then stop; otherwise, go back to Step 2.

The search process in which we consider nonrenewable i is a kind of *variable depth search*; see Kernighan and Lin [1970]. In our search strategy we start from the current best solution and generate a sequence of subsequent neighbors until some stopping criterion is satisfied. Next, a new starting point is chosen for the next sequence. We stop considering nonrenewable i when a maximum number of 100 iterations has been reached, or when all possible starting points have been used. The scaling parameter A, which we have defined on page 70, is set equal to 10.

The output of the algorithm is purchase plan δ'. We refer to this algorithm by

$$\delta' \quad \leftarrow \quad \text{PURCHASE}(\delta, \overline{D}, H).$$

If $\delta = \emptyset$, then the initial solution is determined by using the solution of the linear programming relaxation LP1.

5.1.4 Overview of the algorithm

We start our solution approach by running the algorithm

$$(\delta, \gamma) \quad \leftarrow \quad \text{MINTAR}(\emptyset, \emptyset, 100000).$$

We stop if no feasible solution is obtained. Otherwise, we calculate an appropriate purchase plan by applying the algorithms

$$\begin{aligned} (\gamma, \overline{D}) \quad &\leftarrow \quad \text{DEMAND}(\delta, \gamma); \\ \delta \quad &\leftarrow \quad \text{PURCHASE}(\emptyset, \overline{D}, 10). \end{aligned}$$

We create the final purchase and production plan by scheduling the operations according to activity list γ. Note that the tardiness costs of the solution (δ, γ) are at most equal to the tardiness costs of the production plan that we obtained after running algorithm MINTAR$(\emptyset, \emptyset, 100000)$.

5.2 Ways to improve the integration

In our computational experiments we have incorporated two ways to improve the integration: updating purchase plans during the production planning and backward scheduling.

5.2.1 Updating purchase plans during the production planning

During a run of the algorithm MINTAR$(\delta, \gamma, \overline{I})$ we sometimes obtain schedules that almost improve the current best schedule. After updating the purchase plan the total costs may even be improved. Therefore, we check for such a schedules the possible gain that could be realized by adapting the purchase plan.

Let (δ^*, γ^*) and I be the best solution found during the current run of algorithm MINTAR$(\delta, \gamma, \overline{I})$ and the number of performed iterations, respectively. We denote the value of I at the last improvement by I^* and the tardiness costs of the current best solution by ϕ^*. We adapt the purchase plan for some given solution (δ, γ) if the following three conditions hold.

- activity list γ is feasible,
- $I = I^*$ or $\phi(\delta, \gamma) > \phi(\delta^*, \gamma^*)$, and
- $\phi(\delta, \gamma) + \sum_{u \in \mathcal{E}} w_u d_u \leq (1 + R) e^{\frac{(I^* - I)}{500}} (\phi^* + \sum_{u \in \mathcal{E}} w_u d_u)$, where R is randomly drawn from $U[0, 0.1]$.

To adapt the purchase plan we run the algorithms

$$\begin{aligned} (\gamma, \overline{D}) \quad &\leftarrow \quad \text{DEMAND}(\delta, \gamma); \\ \delta' \quad &\leftarrow \quad \text{PURCHASE}(\emptyset, \overline{D}, 1). \end{aligned}$$

Purchase plan δ' is accepted only if $\chi(\delta', \gamma) < \chi(\delta^*, \gamma^*)$. So running the algorithm results in a solution (δ^*, γ^*). We refer to this algorithm by

$$(\delta^*, \gamma^*) \quad \leftarrow \quad \text{MINTAR+}(\delta, \gamma, \overline{I}).$$

In this version of MINTAR we do not apply restarts: we stop the search process if the best solution has not been improved for 2,500 iterations. Furthermore, a purchase plan δ and a feasible activity list γ are required as input.

5.2.2 Backward scheduling

In the algorithm DEMAND, which we have described in Subsection 5.1.2, we determine for each operation v the latest possible starting time $\overline{S}_v^{\delta\gamma}$ in such a way that

- the tardiness of none of the end-operations increases,
- the processing order on each machine remains the same, and
- the assignment of the produced nonrenewables to the consuming operations remains the same as in solution (δ, γ).

Given these latest possible starting times we determine for each nonrenewable i the demands $\overline{D}_i(t)$ at the beginning of each period t. Therefore, the costs of the new purchase plan depend on the latest starting times. Let weight $\overline{w}_v$ of operation v be equal to the cost of holding the purchased nonrenewables consumed by operation v in inventory for one time unit. A schedule with a higher value of

$$\sum_{v \in \mathcal{V}} \overline{w}_v \overline{S}_v^{\delta\gamma} \tag{5.1}$$

is likely to result in a better purchase plan. In order to increase the value of (5.1) we relax the restriction that the processing order on each machine remains the same.

The resulting latest starting times may be smaller than the corresponding release times. As a consequence the tardiness costs of the new solution may exceed $\phi(\delta, \gamma)$. Therefore, we add to the cost function the penalty

$$\underline{w} \sum_{v \in \mathcal{V}} \max\{0, -\overline{S}_v^{\delta\gamma} + r_v\}, \tag{5.2}$$

where $\underline{w}$ is a given penalty weight.

The purchased nonrenewables that are consumed by operation v must be received before $\overline{S}_v^{\delta\gamma}$. If $\overline{S}_v^{\delta\gamma}$ belongs to period t, we add these nonrenewables to the demands at the beginning of period t. If the last period finishes before time $\overline{S}_v^{\delta\gamma}$,

the required purchased nonrenewables are added to the demands at the beginning of the last period. If the first period starts after time $\overline{S}_v^{\delta\gamma}$, the required purchased nonrenewables are added to the demands at the beginning of the first period. Let $\widetilde{D}_i(t)$ denote the resulting demands. Sometimes these demands cannot be satisfied due to a lack of capacity of the suppliers. In that case a part of the demand is postponed in such a way that they occur as soon as possible. The resulting demand for nonrenewable i at the beginning of period t is denoted by $\overline{D}_i(t)$. Also this modification can result in higher tardiness costs. Therefore, we add a penalty to the cost function given by

$$\underline{w} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} \max\{ \sum_{t' \in \mathcal{T}: t' \leq t} (\widetilde{D}_i(t') - \overline{D}_i(t')), 0\}. \tag{5.3}$$

The total costs are equal to value (5.1) minus penalty (5.2) minus penalty (5.3).

Given a penalty weight $\underline{w}$, we maximize these costs by changing the processing orders on the machines. This is done by a slightly modified version of the tabu search algorithm described in Chapter 3. If the processing order on each machine is given, then there is no reason to advance an operation. Thus finding the optimal latest starting times given an execution order of the operations on each machine can be done using a list scheduling algorithm. Also in this case we use a critical graph to determine the neighborhood. Only in this case everything is reversed. So the root node s succeeds all operations. Since the produced nonrenewables have been assigned to the consuming operations, there are no assignment arcs in the critical graph. Hence, all neighbors are created by swapping pairs of adjacent operations that are processed on the same machine. For every machine arc (u, v) that belongs to a path in the critical graph from s to an operation z for which $\overline{S}_z^{\delta\gamma} < r_z$ or $\overline{w}_z > 0$ we create a neighbor by swapping u and v.

Our algorithm starts with the construction of an activity list, which forms the input of our first iteration. In each iteration we determine the neighborhood $\mathcal{N}(\gamma)$ and use some neighborhood searching strategy to choose a neighbor $\gamma' \in \mathcal{N}(\gamma)$. We stop the search process if the optimum solution has not been improved for 1000 iterations, or if a maximum number of iterations $\overline{I}$ has been reached. Finally, we modify the obtained activity list by ordering all operations according to nondecreasing $\overline{S}_v^{\delta\gamma}$. So running the algorithm results in a new activity list γ' and a list of demands $\overline{D}$. We refer to this algorithm by

$$(\gamma', \overline{D}) \quad \leftarrow \quad \text{MAXSTART}(\delta, \gamma, \underline{w}, \overline{I}).$$

By using a high penalty weight $\underline{w}$ the search is very restricted, but the tardiness costs will never increase. In case of a low penalty weight $\underline{w}$ a higher value of (5.1) is obtained, but the tardiness costs may increase.

5.3 Resulting solution approaches

We have tested four ways to combine the algorithms described in the previous sections. In Section 5.1 we have described a straightforward approach, algorithm ISA1. Now we describe the three other approaches. Hereby, we assume that an appropriate penalty weight $\underline{w}$ has been given. To choose a penalty weight information about the test instances is required.

5.3.1 Solution approach 2

In the integrated solution approach 2 (ISA2) we use the algorithm MAXSTART, which we have described in Subsection 5.2.2. We start with algorithm

$$(\delta, \gamma) \quad \leftarrow \quad \text{MINTAR}(\emptyset, \emptyset, 90000).$$

We stop if no feasible solution is obtained. Otherwise, we run successively the algorithms

$$\begin{aligned} (\gamma, \overline{D}) &\leftarrow \text{MAXSTART}(\delta, \gamma, \underline{w}, 10000); \\ \delta &\leftarrow \text{PURCHASE}(\emptyset, \overline{D}, 10). \end{aligned}$$

5.3.2 Solution approach 3

In the integrated solution approach 3 (ISA3) we alternate several algorithms. The outline of the algorithm is as follows.

Step 1. Run the algorithm

$$(\delta, \gamma) \quad \leftarrow \quad \text{MINTAR}(\emptyset, \emptyset, 45000).$$

Step 2. If no feasible solution is obtained, then stop; otherwise, go to Step 3.

Step 3. Run the algorithms

$$\begin{aligned} (\gamma, \overline{D}) &\leftarrow \text{MAXSTART}(\delta, \gamma, \underline{w}, 5000); \\ \delta &\leftarrow \text{PURCHASE}(\emptyset, \overline{D}, 5). \end{aligned}$$

Step 4. Run the algorithms

$$\begin{aligned} (\delta, \gamma) &\leftarrow \text{MINTAR}(\delta, \gamma, 9000); \\ (\gamma, \overline{D}) &\leftarrow \text{MAXSTART}(\delta, \gamma, \underline{w}, 1000); \\ \delta &\leftarrow \text{PURCHASE}(\delta, \overline{D}, 2). \end{aligned}$$

Step 5. If Step 4 has been performed 5 times, or if the best solution has not been improved in the previous run of Step 4, then stop; otherwise, go back to Step 4.

5.3.3 Solution approach 4

In the integrated solution approach 4 (ISA4) we use the algorithm MINTAR+, which we have described in Subsection 5.2.1. Again we start with algorithm

$$(\delta, \gamma) \quad \leftarrow \quad \text{MINTAR}(\emptyset, \emptyset, 45000).$$

We stop if no feasible solution is obtained. Otherwise, we run successively the algorithms

$$\begin{aligned} (\gamma, \overline{D}) &\leftarrow \text{MAXSTART}(\delta, \gamma, \underline{w}, 5000); \\ \delta &\leftarrow \text{PURCHASE}(\emptyset, \overline{D}, 5); \\ (\delta, \gamma) &\leftarrow \text{MINTAR+}(\delta, \gamma, 50000). \end{aligned}$$

5.4 Computational results

We have tested our algorithms on a randomly generated test sample, which is publically available; see De Bontridder [2001a]. We have generated instances in which none of the nonrenewables are produced by the manufacturer itself. For all instances a trivial lower bound is given. To solve the mixed integer linear programming programs and the linear programming relaxations we used CPLEX, see CPLEX [2000]. We have performed 10 independent runs for each instance. In all experiments the penalty weight $\underline{w}$ was set equal to 1,000,000. The experiments were performed on a Sun Ultra 10 333 MHz workstation.

5.4.1 Test instances

We have generated 32 test instances. In all instances we have 5 suppliers, 5 nonrenewables, 12 periods, 100 operations, 10 end-operations, and 10 machines. For the first 22 instances we have modified the job shop instances that were used by Singer and Pinedo [1998]; Pinedo and Singer [1999]. All of these instances consist of 10 chains of 10 operations each and 10 machines. For all of these instances we have generated an instance of the purchase planning problem as described in Subsection 4.4.1. The input parameter B was set equal to 0.05. To these 22 instances we added nonrenewable resource constraints, due dates and weights. Each operation consumes nonrenewable i with probability 0.2; the consumed quantity is equal to a random integer from $U[1, 100]$. The difficulty of the problem instances is determined by the tightness of the due dates and the relation between the tardiness weights, the inventory costs, and the purchase costs. We have chosen the due dates and the weights in such a way that the resulting problems are interesting from a mathematical point of view. The due dates and the weights are assigned to the last operation of each job, the end-operations. With $\mathcal{J}_e$ we denote the set of operations

that belong to the same job as end-operation e. The due date d_e of end-operation e is

$$\lfloor 1.7 \sum_{u \in \mathcal{J}_e} p_u \rfloor,$$

and the corresponding weight w_e is

$$\lceil (1.3 + R) \sum_{u \in \mathcal{J}_e} \sum_{i \in \mathcal{C}_u} h_i k_u^i / \widetilde{b} \rceil,$$

where R is random number from $U[0, 0.4]$. The resulting instances for our supply chain scheduling problem are denoted by the name of the original job shop instances with suffix 3.

The other 10 instances of our production planning problem have randomly been generated. These instances were generated as follows. First, we have randomly generated a precedence graph using techniques described by Kolisch et al. [1995]. The precedence graph has been generated in such a way that there are 10 end-operations, that 25 operations have no predecessor, and that every operation has at most one direct successor. The resulting problem consists of 10 jobs. The processing time of each operation and the time lag of each precedence relation are random integers from $U[1, 99]$ and $U[0, 19]$, respectively. Each operation is randomly assigned to a machine, and suitable release times and due dates are randomly generated. Also for these instances we have generated a purchase planning problem as described above. The nonrenewable resource constraints and the weights are determined in the same way as for the modified job shop scheduling instances. We denote the resulting instances by bs01–bs10.

5.4.2 Test results

We performed ten independent runs of the algorithms ISA1, ISA2, ISA3 and ISA4 on the test instances and compared the best and the average solutions of both algorithms with a trivial lower bound, which was computed as follows. First, we determine a lower bound for the purchase costs by solving the following instance of the purchase planning problem to optimality. In this instance the inventory costs per period are equal to zero for all nonrenewables and all required nonrenewables are only required at the beginning of the last period. The optimal value of this problem is obtained by solving a mixed integer linear programming problem MIP2 as described in Chapter 3. It is easy to verify that the optimal value is a lower bound for the purchase costs.

Next we determine a lower bound for the sum of the inventory and the purchase costs. In the precedence graph there is only one path from operation v to its end-operation. Let $\bar{l}_v$ denote the length of this path. As mentioned earlier the

		algorithm ISA1			algorithm ISA2		
name	lb	best	average	time	best	average	time
ft10-3	2434324.3	18.33	18.98	65.1	18.33	18.98	62.6
abz5-3	1996499.8	30.47	32.21	1.2	25.47	26.42	44.0
abz6-3	2329692.0	23.44	25.95	1.1	18.83	19.42	29.9
la16-3	1368850.7	18.85	20.88	1.4	17.32	17.53	36.2
la17-3	1513100.4	18.98	20.42	1.7	16.32	17.23	35.7
la18-3	1710800.7	28.22	29.07	1.3	25.13	25.61	39.6
la19-3	1369465.7	24.38	26.25	0.8	21.97	22.75	36.8
la20-3	2275350.9	24.60	25.68	1.0	18.96	19.48	25.8
la21-3	1538974.6	26.86	28.34	1.1	20.70	22.38	33.2
la22-3	3050348.8	22.79	24.90	1.2	19.97	20.38	37.8
la23-3	1814879.6	20.06	21.45	0.9	17.10	17.78	35.7
la24-3	1692463.8	24.14	25.90	1.1	20.86	21.22	31.0
orb01-3	2258817.3	26.17	27.77	69.2	26.17	27.77	65.8
orb02-3	1442598.9	24.26	26.18	2.2	23.51	24.16	24.5
orb03-3	1837348.6	29.68	32.12	64.8	29.64	32.11	63.4
orb04-3	1910843.2	22.01	23.13	5.1	21.79	22.48	27.0
orb05-3	1691513.4	27.96	29.32	51.3	27.96	29.33	53.1
orb06-3	2174932.8	21.45	22.66	16.1	21.06	21.55	55.0
orb07-3	2523152.1	18.09	18.30	1.3	16.32	17.21	37.7
orb08-3	2174471.2	22.25	23.38	76.9	21.86	23.13	87.1
orb09-3	1906651.4	26.03	27.60	5.0	23.94	26.14	30.4
orb10-3	3272011.6	21.43	22.69	3.4	19.74	20.39	29.6
average	2013049.6	23.66	25.14	17.0	21.50	22.43	41.9

Table 5.1: Results per instance.

weight $\overline{w}_v$ of operation v is equal to the cost of holding the purchased nonrenewables consumed by operation v in inventory for one time unit. It is easy to verify that

$$\sum_{v \in \mathcal{V}} \overline{w}_v \bar{l}_v$$

is a weak lower bound for the sum of the inventory and the tardiness costs. The sum of both lower bounds is a valid lower bound for our supply chain scheduling problem.

In Table 5.1 and Table 5.2 we give the results for the modified job shop instances. The results for the instances bs01–bs10 are given in Table 5.3 and Table 5.4. In the second column we give the lower bound (lb). For all algorithms the percentage of deviation of the best and the average solution over the lower bound are given. In the columns 'time' for each instance the average computation time per run is given in cpu seconds.

As a consequence of the weakness of our lower bound, it is difficult to judge the quality of the solutions. Nevertheless, we can conclude that the algorithm MAX-

		algorithm ISA3			algorithm ISA4		
name	lb	best	average	time	best	average	time
ft10-3	2434324.3	18.27	18.56	62.2	14.37	15.29	98.0
abz5-3	1996499.8	25.85	27.17	39.2	23.72	25.14	89.7
abz6-3	2329692.0	18.97	19.73	27.3	17.10	17.81	108.8
la16-3	1368850.7	16.89	17.74	32.1	14.74	15.73	65.8
la17-3	1513100.4	16.32	17.00	33.0	16.07	16.55	40.2
la18-3	1710800.7	25.32	25.70	32.0	22.28	22.95	108.4
la19-3	1369465.7	21.35	22.48	31.3	19.93	20.79	53.5
la20-3	2275350.9	18.96	19.59	26.7	16.37	17.88	62.8
la21-3	1538974.6	20.71	22.53	30.1	19.39	20.53	99.2
la22-3	3050348.8	19.97	20.49	32.1	18.15	18.94	61.8
la23-3	1814879.6	17.10	17.76	31.5	15.81	16.29	64.8
la24-3	1692463.8	20.72	21.40	26.1	18.31	19.40	84.7
orb01-3	2258817.3	25.81	27.60	66.1	23.55	24.73	115.1
orb02-3	1442598.9	23.51	24.40	27.8	20.08	21.47	61.5
orb03-3	1837348.6	29.75	32.06	60.0	27.32	29.35	89.7
orb04-3	1910843.2	21.79	22.48	27.8	18.40	20.15	44.7
orb05-3	1691513.4	27.67	29.17	58.3	22.14	23.71	123.8
orb06-3	2174932.8	21.06	21.83	49.2	17.88	19.07	81.4
orb07-3	2523152.1	16.38	17.27	30.1	15.34	16.36	45.5
orb08-3	2174471.2	22.00	22.97	61.5	19.78	21.61	86.2
orb09-3	1906651.4	23.94	26.48	28.6	24.07	25.15	60.1
orb10-3	3272011.6	19.74	20.54	29.3	18.57	19.46	61.9
average	2013049.6	21.46	22.50	38.3	19.24	20.38	77.6

Table 5.2: Results per instance.

START has a positive effect on the quality of the solutions. The quality of the solutions of algorithms ISA2 and ISA3 is almost the same, but the quality of the solutions of algorithm ISA4 is better.

5.5 Conclusion

In this chapter we have presented several solution approaches for our supply chain scheduling problem. We tested these approaches on several test instances. During our experiments we noted that the results highly depend on the type of problem instances. So the chosen solution approach depends on the relation between the tardiness costs, the inventory costs, and the purchase costs. The determination of appropriate weighting factors for different costs is one of the main difficulties of supply chain scheduling problems. Nevertheless, we can conclude that it is useful to integrate purchase and production planning.

As in many supply chain scheduling problems it appears that solving both subproblems simultaneously gives the best results. This can easily be accomplished,

		algorithm ISA1			algorithm ISA2		
name	lb	best	average	time	best	average	time
bs01-3	2056234.7	15.50	16.68	0.9	11.55	11.71	35.6
bs02-3	2277500.9	17.27	18.00	0.7	14.86	15.20	38.8
bs03-3	2994444.1	10.48	11.06	41.1	10.48	11.06	41.1
bs04-3	1892658.7	18.03	19.00	0.9	14.22	15.25	46.5
bs05-3	3276393.2	19.70	20.76	1.3	15.60	15.92	39.4
bs06-3	2981442.0	12.59	12.99	1.2	9.92	10.16	50.1
bs07-3	1900203.4	12.40	13.48	0.8	9.88	10.81	47.2
bs08-3	2242311.5	12.48	13.83	0.8	8.47	9.18	38.1
bs09-3	2158115.4	19.17	19.76	46.4	19.17	19.76	46.5
bs10-3	2726096.2	17.59	18.11	65.0	16.76	17.96	72.6
average	2450540.0	15.52	16.37	15.9	13.09	13.70	45.6

Table 5.3: Results per instance.

		algorithm ISA3			algorithm ISA4		
name	lb	best	average	time	best	average	time
bs01-3	2056234.7	11.55	11.85	32.8	9.70	10.21	138.0
bs02-3	2277500.9	14.78	15.20	37.7	13.23	13.99	65.9
bs03-3	2994444.1	9.37	9.46	64.1	8.47	8.67	221.3
bs04-3	1892658.7	14.22	15.26	37.6	13.45	14.05	74.4
bs05-3	3276393.2	15.60	15.92	35.0	13.82	15.03	80.6
bs06-3	2981442.0	9.92	10.22	41.7	9.11	9.56	107.2
bs07-3	1900203.4	9.88	10.75	36.8	9.09	9.83	57.8
bs08-3	2242311.5	8.47	9.18	32.9	7.89	8.56	75.5
bs09-3	2158115.4	17.14	17.43	72.1	14.28	15.31	96.7
bs10-3	2726096.2	16.76	17.01	69.8	16.49	16.84	76.6
average	2450540.0	12.77	13.23	46.1	11.55	12.20	99.4

Table 5.4: Results per instance.

since both subproblems are solved through local search approaches. In order to integrate both local search algorithms we have to make small changes to the solutions of the different subproblems in some appropriate order. To handle other supply chain scheduling problems the same straightforward approach can be used. First we decompose the problem into appropriate subproblems. After that, we develop for these subproblems local search algorithms. Hereby it is useful to use mathematical programming techniques to incorporate problem-specific knowledge. Finally, the local search algorithms are integrated to one solution approach.

A very restrictive factor in our supply chain scheduling problem is the assumption that the total purchased quantity of each nonrenewable is given. Dropping this assumption results is an interesting problem. In this situation the manufacturer can

purchase extra nonrenewables in order to obtain a better production plan. This problem can even be extended further by allowing modifications of the set of operations, i.e., extra operations that produce certain nonrenewables could be added in order to reduce the purchase costs, or operations could be removed in order to reduce the tardiness costs.

Notice that we have only mentioned some extensions of our supply chain scheduling problem. But in supply chains there are many other problems that could be considered in an integrated context. So there are many interesting challenges in the area of supply chain scheduling.

Bibliography

B.J.M. AARTS [1996]. *A parallel local search algorithm for the job shop scheduling problem*, Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

E.H.L. AARTS AND J.K. LENSTRA (EDITORS) [1997]. *Local Search in Combinatorial Optimization*, Wiley, Chichester.

J. ADAMS, E. BALAS, AND D. ZAWACK [1988]. The shifting bottleneck procedure for job shop scheduling, *Management Science* 34, 391–401.

A. AGGARWAL AND J.K. PARK [1993]. Improved algorithms for economic lot size problems, *Operations Research* 41, 549–571.

E.H. AGHEZZAF AND L.A. WOLSEY [1994]. Modelling piecewise linear concave costs in a tree partitioning problem, *Discrete Applied Mathematics* 50, 101–109.

R.K. AHUJA, T.L. MAGNANTI, AND J.B. ORLIN [1993]. *Network Flows – Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, New Jersey.

D. APPLEGATE AND W. COOK [1991]. A computational study of the job shop scheduling problem, *ORSA Journal on Computing* 3, 149–156.

T. BAAR, P. BRUCKER, AND S. KNUST [1998]. Tabu-search algorithms for the resource-constrained project scheduling problem, In S. Voss, S. Martello, I. Osman, and C. Roucairol (editors), *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 1–18, Kluwer Academic Publishers, Boston.

H.C. BAHL, L.P. RITZMAN, AND J.N.D. GUPTA [1987]. Determining lot sizes and resource requirement: a review, *Operations Research* 35, 329–345.

E. BALAS [1969]. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, *Operations Research* 17, 941–957.

M.S. BAZARAA, J.J. JARVIS, AND H.D. SHERALI [1990]. *Linear Programming and Network Flows, 2nd ed.*, Wiley, New York.

P. BRUCKER, A. DREXL, R. MÖHRING, K. NEUMANN, AND E. PESCH [1999]. Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* 112, 3–41.

J. CARLIER AND A.H.G. RINNOOY KAN [1982]. Scheduling subject to nonrenewable-resource constraints, *Operations Research Letters* 1, 52–55.

V. ČERNÝ [1985]. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, *Journal of Optimization Theory and Applications* 45, 41–51.

P. COWLING AND W. REZIG [2000]. Integration of continuous caster and hot strip mill planning for steel production, *Journal of Scheduling* 3, 185–208.

CPLEX [2000]. *CPLEX 6.6 Documentation Supplement*, ILOG CPLEX Division, Incline Village, Nevada.

K.M.J. DE BONTRIDDER [2000a]. A generalized job shop with nonrenewable resource constraints (test instances), *www.win.tue.nl/~koendb/*.

K.M.J. DE BONTRIDDER [2000b]. Minimizing total weighted tardiness in a generalized job shop (test instances), *www.win.tue.nl/~koendb/*.

K.M.J. DE BONTRIDDER [2001a]. Integrating purchase and production planning (test instances), *www.win.tue.nl/~koendb/*.

K.M.J. DE BONTRIDDER [2001b]. A purchase lot sizing problem with quantity discounts (test instances), *www.win.tue.nl/~koendb/*.

B. FAALAND AND T. SCHMITT [1987]. Scheduling tasks with due dates in a fabrication/assembly process, *Operations Research* 35, 378–388.

H. FISHER AND G.L. THOMPSON [1963]. Probabilistic learning combinations of local job-shop scheduling rules, In J.F. Muth and G.L. Thompson (editors), *Industrial Scheduling*, pages 225–251, Prentice Hall, Englewood Cliffs, New Jersey.

M.R. GAREY AND D.S. JOHNSON [1979]. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

F. GLOVER [1989]. Tabu search: part 1, *ORSA Journal on Computing* 1, 190–206.

F. GLOVER [1990]. Tabu search: part 2, *ORSA Journal on Computing* 2, 4–32.

J. HABERL [1999]. Fixed-charge continuous knapsack problems and pseudogreedy solutions, *Mathematical Programming* 85, 617–642.

W. HERROELEN, E. DEMEULEMEESTER, AND B. DE REYCK [1998]. A classification scheme for project scheduling, In J. Weglarz (editor), *Project Scheduling : Recent Models, Algorithms and Applications*, volume 14 of *International Series in Operations Research and Management Science*, pages 1–26, Kluwer Academic Publishers, Boston.

J.A. HOOGEVEEN, J.K. LENSTRA, AND S.L. VAN DE VELDE [1997]. Sequencing and scheduling, In M. Dell'Amico, F. Maffioli, and S. Martello (editors), *Annotated Bibliographies in Combinatorial Optimization*, Wiley, Chichester.

B.W. KERNIGHAN AND S. LIN [1970]. An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal* 49, 291–307.

S. KIRKPATRICK, JR. C.D. GELATT, AND M.P. VECCHI [1983]. Optimization by simulated annealing, *Science* 220, 671–680.

R. KOLISCH AND K. HESS [2000]. Efficient methods for scheduling make-to-order assemblies under resource, assemble area, and part availability constraints, *International Journal of Production Research* 38, 207–228.

R. KOLISCH, A. SPRECHER, AND A. DREXL [1995]. Characterization and generation of resource-constrained project scheduling problems, *Management Science* 41, 1693–1703.

S. KREIPL [2000]. A large step random walk for minimizing total weighted tardiness in a job shop, *Journal of Scheduling* 3, 125–138.

P.J.M. VAN LAARHOVEN, E.H.L. AARTS, AND J.K. LENSTRA [1992]. Job shop scheduling by simulated annealing, *Operations Research* 40, 113–125.

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS [1993]. Sequencing and scheduling: Algorithms and complexity, In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (editors), *Logistics of Production and Inventory*, volume 4 of *Handbooks in OR & MS*, pages 445–522, Elsevier, Amsterdam.

S. LAWRENCE [1984]. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement), Technical report, Graduate School of Industrial Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania.

M. MASTROLILLI AND L.M. GAMBARDELLA [2000]. Effective neighborhood functions for the flexible job shop problem, *Journal of Scheduling* 3, 3–20.

E. NOWICKI AND C. SMUTNICKI [1996]. A fast taboo search algorithm for the job shop problem, *Management Science* 42, 797–913.

M. PINEDO [1995]. *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, New Jersey.

M. PINEDO AND M. SINGER [1999]. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop, *Naval Research Logistics* 46, 1–19.

B. ROY AND B. SUSSMANN [1964]. Les problèmes d'ordonnancement avec contraintes disjonctives, Note DS No. 9 bis, SEMA, Montrouge.

P.A. RUBIN AND W.C. BENTON [1993]. Jointly constrained order quantities with all-units discounts, *Naval Research Logistics* 40, 255–278.

D. SIMCHI-LEVI, P. KAMINSKI, AND E. SIMCHI-LEVI [2000]. *Designing and Managing the Supply Chain: Concepts, Strategies, and Cases*, Irwin/McGraw-Hill, New York.

M. SINGER AND M. PINEDO [1998]. A computational study of branch and bound techniques for minimizing the total weighted tardiness in flow shops and job shops, *IIE Scheduling and Logistics* 29, 109–119.

W.E. SMITH [1956]. Various optimizers for single stage production, *Naval Research Logistics Quarterly* 3, 59–66.

H. STADLER AND C. KILGER (EDITORS) [2000]. *Supply Chain Management and Advanced Planning Systems: Concepts, Models, Software and Case Studies*, Springer, Heidelberg.

STANDARD PERFORMANCE EVALUATION CORPORATION [1999]. Performance results specint95, *www.spec.org/cgi-bin/osgresults?conf=cint95*.

E.D. TAILLARD [1994]. Parallel taboo search techniques for the job shop scheduling problem, *ORSA Journal on Computing* 6, 108–117.

G.J.A. TEEUWEN [1999]. *Operational Planning of Supply Chains with an Application in the Newspaper Industry*, Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

R.J.M. VAESSENS, E.H.L. AARTS, AND J.K. LENSTRA [1996]. Job shop scheduling by local search, *INFORMS Journal on Computing* 8, 302–317.

M.G.A. VERHOEVEN [1998]. Tabu search for resource-constrained scheduling, *European Journal of Operational Research* 106, 266–276.

M. WENNINK [1995]. *Algorithm Support for Automated Planning Boards*, Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

M. WENNINK [2000]. Finding selection-optimal schedules by means of maximum cost flows, *Operations Research Letters*, to appear.

M. WENNINK AND R. VAESSENS [1995]. An efficient insertion algorithm with multiprocessor operations, Working paper, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Samenvatting

Door de sterke ontwikkeling van informatiesystemen zoals ERP, is er in de afgelopen jaren een steeds grotere interesse ontstaan voor planningsproblemen die voorkomen in supply chains. Door de invoering van ERP systemen komen alle data uit de supply chain beschikbaar op één lokatie. Hierdoor kunnen problemen die vroeger apart werden bekeken nu op een geïntegreerde manier worden aangepakt. Het doel van supply chain scheduling is het realiseren van de voordelen die hierdoor mogelijk zijn. In dit proefschrift behandelen we een voorbeeld van een supply chain scheduling probleem, waarin we het aankoop- en het productieplan van een fabrikant op een geïntegreerde manier bepalen. We kijken naar een fabrikant die producten maakt voor klanten. Deze producten moeten worden geleverd op een afgesproken tijdstip. Indien dit tijdstip niet gehaald wordt, moet de fabrikant een boete betalen. Deze boetes hangen lineair af van de hoeveelheid vertraging.

Het productieproces van ieder product bestaat uit één of meer stappen. In het productieplan wordt het starttijdstip voor elke stap in het productieproces vastgelegd. Zo'n stap in het productieproces moet worden uitgevoerd op een vooraf bepaalde machine, die maar één opdracht tegelijkertijd kan uitvoeren. Deze capaciteitsbeperking is een van de voornaamste restricties waar we rekening mee moeten houden bij het bepalen van een productieplan. Een andere belangrijke restrictie is dat een stap in het productieproces pas kan worden uitgevoerd als alle benodigde materialen beschikbaar zijn. Deze materialen worden geleverd door verscheidene externe leveranciers of geproduceerd door de fabrikant zelf.

In het aankoopplan worden de leveranciers, de levertijdstippen en de te leveren hoeveelheden van de te leveren materialen vastgelegd. Met alle leveranciers zijn prijzen, kortingen en andere voorwaarden overeengekomen. Bij het bepalen van een aankoopplan moet de fabrikant twee zaken tegen elkaar afwegen. Door veel materiaal tegelijkertijd te bestellen kan de fabrikant optimaal gebruik maken van de door de leverancier aangeboden kortingen en zo de totale aankoopprijs drukken. Aan de andere kant kunnen grote bestellingen tot grotere voorraadkosten leiden, omdat het materiaal (al dan niet verwerkt) langer dan noodzakelijk in het productieproces is. Het doel van de fabrikant is om een aankoop- en een productieplan te bepalen zodanig dat de som van de aankoopkosten, productiekosten en de boetes veroorzaakt door een te late levering zo laag mogelijk zijn.

In dit proefschrift beschrijven we eerst algoritmes voor het oplossen van drie deelproblemen van dit supply chain scheduling probleem. Daarna beschrijven we een methode om al deze algoritmes te combineren tot een geïntegreerde aanpak voor ons supply chain scheduling probleem. Bij de eerste twee deelproblemen gaan we ervan uit dat het aankoopplan gegeven is en dat dus alleen het productieplan bepaald moet worden. Het verschil tussen beide deelproblemen is dat in het eerste geval voor de geproduceerde of geleverde materialen al exact bepaald is waarvoor ze zullen worden gebruikt. In het tweede deelprobleem is dit niet het geval, waardoor we meer vrijheid hebben bij het bepalen van het productieplan. In het derde deelprobleem gaan we ervan uit dat het productieplan al vastligt en dat we alleen het aankoopplan nog moeten bepalen.

In al onze algoritmes maken we gebruik van zogenaamde lokale zoektechnieken. Het basis idee van lokale zoektechnieken is dat we een gegeven oplossing kunnen verbeteren door kleine veranderingen aan te brengen. Door te beginnen bij een startoplossing en het herhaaldelijk aanbrengen van kleine veranderingen proberen we steeds betere oplossingen te vinden. Het is van belang om bij het ontwikkelen van een lokaal zoekalgoritme gebruik te maken van probleem-specifieke informatie. Om dit te realiseren gebruiken we in al onze algoritmes technieken uit de discrete optimalisering.

Acknowledgments

This thesis results from a research project which was financed by Baan and carried out at the Department of Mathematics and Computer Science of the Technische Universiteit Eindhoven. During the first years part of the research was done at the supply chain solutions group of Baan.

First of all I thank Baan for supporting the research. During the entire project Marc Sol was my advisor at Baan. Marc was always very enthusiastic, and I am indebted to him for many useful ideas. Furthermore, I express my gratitude to Jaco van Kooten and Paul Giesberts for initiating the project and to Marco Verhoeven, Martin Taal and all other members of the former supply chain solutions group in Ede for their interest and support.

Second, I wish to thank Han Hoogeveen. Since the beginning, he has been actively involved in both research and the process of writing the results down. Thanks to his efforts the results and the readability have improved a lot. Furthermore, I thank my supervisors Jan Karel Lenstra and Emile Aarts for their advice and support. Their suggestions and comments have always been very valuable. I am also grateful to the other members in my dissertation committee, Mike Pinedo and Steef van de Velde, for providing useful comments, and to Marc Wennink for providing information about his research.

Finally, I thank my colleagues of the section operations research and statistics of the Technische Universiteit Eindhoven for making it such a pleasant place to work and my family and friends for their continuous support and encouragement.

Curriculum vitae

Koen De Bontridder was born on November 26th, 1973, in Heel en Panheel, The Netherlands. In 1992, he received his Atheneum diploma (pre-university level) from the Sint Maartens College, Maastricht. In the same year he started to study industrial and applied mathematics at the Eindhoven University of Technology. After the first year he decided to specialize in operations research and statistics. He graduated with honors in 1997 after having completed his Master's thesis entitled 'Methods for Solving the Test Cover Problem'. In June 1997 he started as a PhD-student at the Eindhoven University of Technology under the supervision of professor dr. J.K. Lenstra, professor dr. E.H.L. Aarts, dr. J.A. Hoogeveen, and dr. ir. M. Sol. His research was focussed on the development and analysis of models and algorithms for planning and scheduling problems that arise in supply chains. Part of the research was done at the Supply Chain Solutions Group of Baan. The results are presented in this PhD-thesis.

Stellingen

behorende bij het proefschrift

Integrating Purchase and Production Planning

Using Local Search in Supply Chain Optimization

van

Koen De Bontridder

I

Beschouw het volgende probleem. Gegeven een graaf, bepaal het maximale aantal knoop-disjuncte paden van lengte 2. Voor dit probleem definiëren we een serie iteratieve verbeteringsalgoritmen H_k ($k \geq 0$). Algoritme H_k probeert in iedere stap het aantal paden van lengte 2 met tenminste één te laten toenemen. Om dit te bereiken mogen maximaal k paden van lengte 2 verwijderd worden. Het algoritme stopt als op deze manier geen verbetering meer mogelijk is. De prestatiegarantie ρ_k van algoritme H_k voor $k = 0, 1, 2, 3, 4$ is als volgt:

k	0	1	2	3	4
ρ_k	3	2	$\frac{9}{5}$	$\frac{11}{7}$	$\frac{3}{2}$

K.M.J. DE BONTRIDDER, B.V. HALLDÓRSSON, M.M. HALLDÓRSSON, C.A.J. HURKENS, J.K. LENSTRA, R. RAVI, AND L. STOUGIE [2001], Approximation algorithms for the minimum test set problem, Manuscript.

II

Beschouw het volgende probleem. Gegeven zijn m objecten en n tests. Ieder object reageert positief of negatief op een test. Het doel is om het minimale aantal tests te bepalen dat nodig is om alle objecten te kunnen onderscheiden. Twee objecten zitten in één equivalentieklasse dan en slechts dan als ze niet te onderscheiden zijn door de al uitgevoerde tests. Stel dat er na het uitvoeren van een aantal tests k equivalentieklassen $E_1, \ldots, E_k$ overblijven. Om de nog niet uitgevoerde tests te vergelijken kunnen onder andere het onderscheidingsvermogen en de informatiewaarde gebruikt worden. Het onderscheidingsvermogen van een test T is gelijk aan

$$\sum_{i=1}^{k} \min\{|T \cap E_i|, |E_i \setminus T|\}.$$

De informatiewaarde is gelijk aan

$$\log_2 m - \frac{1}{m} \sum_{i=1}^{k} |E_i| \log_2 |E_i|.$$

Uit rekenresulaten blijkt dat branch-and-bound algoritmen ter bepaling van het minimale aantal tests beter gebaseerd kunnen worden op het onderscheidingsvermogen dan op de informatiewaarde.

K.M.J. DE BONTRIDDER, C.A.J. HURKENS, J.K. LENSTRA, J.B. ORLIN, AND L. STOUGIE [2001]. Branch and bound algorithms for the minimum test set problem, Manuscript.

III

Het gebruik van probleemspecifieke informatie in lokale-zoekalgoritmen is zinvol.

IV

Door lokale-zoekalgoritmen volgens de standaardregels uit de literatuur toe te passen worden de mogelijkheden onnodig beperkt.

V

Vanwege de volgende omslachtige spellingregel is het in Nederland vaak onmogelijk om persoonsnamen correct te laten schrijven.

Persoonsnamen krijgen een hoofdletter. Het voorzetsel of lidwoord krijgt een hoofdletter als er geen naam of voorletter aan voorafgaat. In Vlaanderen behouden lidwoorden en voorzetsels van persoonsnamen altijd hun originele schrijfwijze.[1]

Het verdient daarom aanbeveling om de Vlaamse schrijfwijze tot standaard te maken.

1 INSTITUUT VOOR NEDERLANDSE LEXICOLOGIE [1995], *Woordenlijst Nederlandse taal*, Sdu Uitgevers, Den Haag.

VI

Gezien het programma van de Nederlandse voetbalcompetitie is het merkwaardig dat champions league poules met Nederlandse deelname wel een regulier verloop kennen.

VII

Dat gemeenschapsonderdanen zich in Nederland eigenlijk nog altijd moeten melden bij de vreemdelingendienst wijst op een gebrekkige Europese integratie.

In het EG-verdrag is de bepaling opgenomen dat de gemeenschapsonderdaan zijn verblijf dient aan te melden bij het bevoegd gezag en dat de gemeenschapsonderdaan in het bezit kan worden gesteld van een verblijfsdocument. Deze meldingsplicht en de sancties op het zich niet houden daaraan, mogen echter niet leiden tot een belemmering van het vrij verkeer van personen. Daarom hoeft u zich niet te melden bij de Vreemdelingendienst. Wij raden u echter aan om u wel te melden bij de vreemdelingendienst, om eventuele problemen met andere overheidsinstanties te voorkomen.[2]

2 IMMIGRATIE- EN NATURALISATIEDIENST [2001], *De toelating van gemeenschapsonderdanen tot Nederland*, Ministerie van Justitie, Den Haag.

VIII

De omslag van een proefschrift geeft een goed beeld van de promovendus.

IX

Wetenschappelijk onderzoekers krijgen hun beste inspiratie niet op hun werkplek, maar bijvoorbeeld in berghutten of treinen.

X

Tentoonstellingsmakers scheppen condities, af en toe kunnen ze het geheel dirigeren maar de kunst blijft de hoofdzaak. Die kunst zal zich wel richten naar zijn ruimtelijke/tijdelijke context, maar die kunst spiegelt zich vooral aan de andere kunst; niet omwille van zichzelf maar omwille van zijn toeschouwers. De toeschouwer is de context.

G.P. DE BONTRIDDER [1986]. Context, *Het Bassin*, jaargang 2, nummer 2, 46–47.