UNIVERSITÄT OSNABRÜCK

Dissertation

Integrated Production and Distribution Scheduling

Dipl.-Inform.
Christian Viergutz

Schriftliche Arbeit zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
im Fach Informatik

eingereicht im
Fachbereich Mathematik / Informatik
Universität Osnabrück

Osnabrück, im Februar 2011

Gutachter:
Prof. Dr. Sigrid Knust, Universität Osnabrück, Deutschland
Prof. Chris N. Potts, University of Southampton, Großbritannien

Mitglieder der Prüfungskommission:
Prof. Dr. Sigrid Knust
Prof. Dr. Oliver Vornberger
Prof. Dr. Volker Sperschneider
Dr. Jutta Göers

Tag der Disputation: 23. Juni 2011

Danksagung

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Institut für Informatik der Universität Osnabrück entstanden. An dieser Stelle möchte ich mich nun bei allen bedanken, die auf verschiedene Weise dazu beigetragen haben.

Für die ausgezeichnete Betreuung bedanke ich mich zunächst ganz besonders bei Prof. Dr. Sigrid Knust, mit der ich über 4 Jahre lang zusammenarbeiten durfte, und die mich auf das Thema meiner Dissertation aufmerksam gemacht hat. Ihre wertvollen Anregungen und Hinweise in gemeinsamen Diskussionen haben einen unschätzbaren Anteil am Gelingen dieser Arbeit.

Mein Dank gilt zudem Florian Bruns und Rolf Wendt, die mir in fachlichen Gesprächen immer wieder Impulse und Tipps gegeben haben.

Weiterhin möchte ich mich bei meinen Kolleginnen und Kollegen im Institut für Informatik für die gute Zusammenarbeit und die angenehme Arbeitsatmosphäre bedanken, speziell bei Friedhelm Hofmeyer für seine technische Unterstützung.

Ich danke allen, die mir beim Korrekturlesen meiner Dissertation geholfen haben, insbesondere meiner Schwester Natalie und Florian Bruns.

Herzlichen Dank sage ich meinen Eltern und Geschwistern für ihren Rückhalt und ihre Unterstützung ganz besonders während meiner Promotionszeit.

Schließlich möchte ich meiner Freundin Kirstin Spenner herzlich dafür danken, dass sie mir immer wieder Aufmunterung und Ansporn bei der Erstellung dieser Arbeit gegeben hat.

Vielen herzlichen Dank!

Osnabrück, im Februar 2011 Christian Viergutz

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

The increasing importance of methods in Operations Research (OR) since the 1950s is largely based on the continuing progress and achievements obtained particularly in the area of production and distribution planning. Industrial and logistics companies have always attempted to increase the performance and flexibility of their production and delivery processes in order to extend their revenues and reduce costs. The ongoing research in Supply Chain Management (SCM) has contributed to the development of planning procedures and tools that are capable of handling complex supply chain networks. Facing the needs caused by increasing globalization and fierce competition in many industries, companies are not only forced to compete with others with regard to pricing or product quality, but also with regard to reliability and timeliness of the deliveries.

Multi-stage supply chains generally pose the difficulty of insufficient coordination between the individual stages. This incurs a loss of efficiency, which in turn almost inevitably leads to competitive disadvantages that may be crucial to the business of a company. Consequently, firms must strive to integrate the planning processes of several stages in their supply chains.

Since the planning and scheduling processes involved in the production and subsequent distribution are known for their intricacy and complexity, these planning problems have traditionally been treated as independent. This may be regarded as an appropriate assumption if those activities are sufficiently decoupled by stock-keeping. However, in several branches of industry where, for example, products with a short lifespan or

highly-customized products are manufactured, keeping inventories in a large scale for a long period is not a preferable option, either because the commodities are subject to expiration and decay or because the customer satisfaction is negatively influenced by long periods between order and delivery times. Thus, an integrated planning approach is advantageous, respecting the interdependencies of production and transportation scheduling.

The enormous technological progress in computing power and the ubiquitous means of communication provided by the Intenet and mobile devices have enabled the – at least *near*-optimal – solution of computationally complex planning problems that seemed impossible to solve only a few decades ago. Nevertheless, many practical problems in SCM need simplifying assumptions when it comes to model formulation in order to make them computationally tractable for larger problem sizes. This means that these models have to leave out some practical restrictions that should, of course, rather be obeyed. Thus, a long-term goal is to eventually integrate these requirements into the scheduling models, which, in this case, are often referred to as "rich" variants of classical scheduling models. Besides their $\mathcal{NP}$-hardness, which means that finding efficient (polynomial-time) algorithms for their solution appears very unlikely, it is already very difficult to compute optimal solutions even for surprisingly small or moderate-sized problem instances. Since also larger instances need to be solved in practice, efficient heuristic methods must be developed that generally cannot guarantee to find an optimal solution, but a near-optimal one within a reasonable runtime.

Scope and objectives of this thesis

In the thesis at hand we consider integrated production and distribution scheduling problems, which frequently occur in the manufacturing of products with a short lifespan or in make-to-order business models where the production is triggered only if a corresponding customer order is received. We will focus on coordinating production and distribution processes in such a way that the customers' requirements will be met as closely as possible while at the same time the sum of revenues earned by the manufacturer is maximized.

The basic problem scenario that will shift into the center of our interest is mainly inspired by a production and distribution model that was introduced by Armstrong et al. (2008). Their research covers the treatment of a common subproblem encountered in two-stage supply chains. Due to these subproblem characteristics, their model abstracts from practically relevant restrictions and (necessarily) makes simplifying assumptions. However, this core model provides us with an interesting starting point and a solid foundation for further practical extensions that we will discuss and

implement in this thesis. We will present a classification of some important variants of the problem and introduce corresponding mathematical programming formulations. An objective spanning over the entire thesis consists in coping with the tremendous complexity of the problems by providing appropriate heuristic solution algorithms which are designed to obtain high quality solutions quickly. We will put our focus on the development of these heuristics and on the evaluation and comparison of their respective performance.

Applications

Practical applications in which the previously mentioned problems need to be solved include the production and transportation of certain intermediate products like adhesive materials in chemical industry. The corresponding planning problem is being complicated by the fact that the considered chemicals have a short lifespan of only a few hours when being transported, but substantially longer when they are kept in a special storage at the premises of the customers. Furthermore, in order to reduce the required storage space at both the producer and the consumers, the delivery times of the adhesives should meet the production plans of the customers, which is included in the model by customer-specified delivery time windows.

Another application concerns the delivery of ready-mix concrete from a production plant to several construction sites by a fleet of delivery trucks. The set of possible tours for these vehicles is restricted by the limited time during which the concrete stays workable in the truck compartments. In contrast to other production and distribution scenarios, in ready-mix concrete delivery, a single truck usually serves only a small number of customers – if it serves more than one at all. Furthermore, the total demand of a customer during the planning period can typically not be satisfied by a single delivery. However, the models that we consider in this thesis will assume that each customer is visited at most once per planning period.

As mentioned before, integrated production and distribution models can also be applied in make-to-order manufacturing settings, where it is common that highly customized products are produced in direct response to specific customer orders. For example, some companies that build personal computers according to the requirements of customers have adopted this business model. But also service providers working in the final customization stages of home textiles frequently fall into this category. Many firms in this industry have outsourced the manufacturing of their products (e.g. window curtains or beddings) but retain the final customization (e.g. sizing or labeling) at local distribution centers. Clearly, there is virtually no finished-goods inventory for such customized textiles.

1.2 Outline

This thesis is composed of seven chapters which are organized as follows. The first two chapters (including this introduction) focus on foundational aspects, such as the presentation of relevant literature and a basic integrated production and distribution model that is of fundamental importance for the remainder of this work. In the following three chapters (3–5), we dicuss our extensions and develop solution algorithms for several problem variants. The final two chapters contain the results of our computational experiments, analyze the performance of the presented algorithms and draw conclusions. A graphical outline that represents the overall structure of this thesis is shown in Figure 1.1.

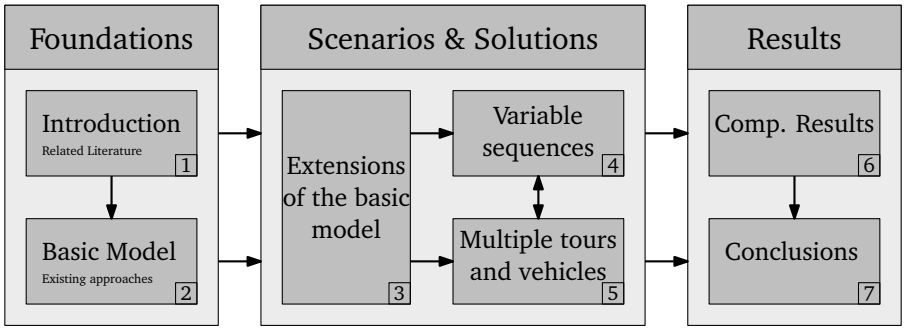


Figure 1.1: A graphical outline of this thesis

In Chapter 2 we present a detailed description of a basic integrated production and distribution scheduling model initially proposed by Armstrong et al. (2008) for a single vehicle that is responsible for delivering a perishable product from a production facility to a set of geographically dispersed customers on a single tour. The customers place orders associated with a demand of the product during the planning period and specify time windows for the deliveries. Since these restrictions are considered as requirements that must be fulfilled for admissible deliveries, possibly not all customer demands can be satisfied within the planning period. As a consequence, the manufacturing company has to make a decision which orders to accept or reject. We give a formal definition of this model and point out the existing approaches to solve the problem. Furthermore, the chapter serves to establish a common notation (for model parameters, variables, etc.) which is used throughout the thesis.

Chapter 3 discusses various extensions of the basic model which are motivated by the fact that this model is quite constrained and relies on many simplifying assumptions. It is our goal to incorporate more realistic restrictions from practical applications. Moreover, we point out some corrections and improvements for the algorithms proposed

by Armstrong et al. In addition to that, we address the issue of finding feasible start times for the production and distribution phases, which is necessary due to the limited lifespan of the considered product. We implement these ideas in a metaheuristic approach based on tabu search that has proven to be a good alternative for exact solution methods presented previously.

Chapter 4 is devoted to more rigorous extensions of the basic model, where variable production and distribution sequences are introduced. On the one hand, these generalizations lead to far more realistic model variants and better solutions, while on the other hand, however, they come at the price of a significant increase in problem complexity. Thus, we concentrate on the development of fast heuristic methods to solve these combined selection and sequencing problems.

In Chapter 5 we make the transition from the single tour scenario, where only one vehicle tour must be planned, to the multi-tour or, even more general, the multi-vehicle scenario, which permits the utilization of several vehicles to execute the deliveries. Moreover, each single vehicle is allowed to perform intermediate returns to the production plant, in which case the route of the vehicle consists of multiple tours during the planning period. Since this scenario represents a further generalization of the model described in Chapter 4, there is little hope of finding efficient exact solution algorithms. Besides presenting a comprehensive mathematical programming formulation for this most general problem scenario that we consider, we also study heuristic solution approaches based on local search in order to tackle the given challenges.

The results of our computational experiments can be found in Chapter 6. We have conducted an analysis and a survey of these results in order to evaluate the performance of the proposed algorithms. As already mentioned, most of the algorithms are implementations of heuristic solution strategies. Furthermore, since appropriate problem instances are not available for the model variants that we study, we have implemented a program to generate suitable random instances. The chapter contains a description of how the generation process works and which key parameters are involved.

Finally, in Chapter 7 we conclude this thesis by summarizing our main results. Moreover, we give an outlook by pointing out topics for future research.

In the following section we present a detailed overview of literature which is related to the problems that we study. Since problems from the areas of production and transportation planning are involved and also solution techniques from combinatorial optimization are discussed, the section is divided into groups of similar topics.

1.3 Survey of related literature

The topic of integrated production and distribution scheduling has been discussed with rapidly growing intensity in the literature, especially during the last 5 years. In many applications it has been shown that a closely combined planning of the production and delivery stages can substantially reduce costs or increase the level of service for customers.

In this section we give a survey of related literature, where most of the referenced articles involve the presentation of applications, mathematical models and solution approaches, such as dedicated heuristics or exact algorithms, together with a presentation of computational results and performance comparisons. The most important topics covered by the given references are summarized in the following.

Make-to-order Manufacturing In *make-to-order* manufacturing, the production of goods is triggered by incoming orders that might prescribe which possible variant(s) of a product should be produced. Typically, there is only a small amount of finished goods inventory in the system, waiting to be shipped to the customers. This scenario is common for production with a short lifespan or for highly customized products. Examples include some vendors of personal computers whose components are specified at order time or also car production where customers are allowed to choose from a list of optional equipment for a given basic car model.

By contrast, in *make-to-stock* manufacturing, the production is generally independent from specific orders. The aim is to produce up to a certain level of finished goods inventory, which is typically ordered by retailers. This scenario is often found in mass production, which involves standardized products that do not need further customization.

Vehicle Routing The Vehicle Routing Problem (VRP) is an important problem in combinatorial optimization, where a number of geographically dispersed customers must be served by a fleet of vehicles (typically from a central depot) so that the total cost of distribution is minimized. The VRP contains the well-known Traveling Salesman Problem (TSP) as a special case, where only one vehicle is available for the deliveries. The problem was first described by Dantzig and Ramser (1959) as the "Truck Dispatching Problem" and has since been extended in several ways (see Toth and Vigo, 2002), for example by introducing delivery time windows for the customers (VRP with time windows, VRPTW) or requiring the vehicles not only to deliver but also to collect goods and transfer them to drop-off locations (VRP with pickup and delivery, VRPPD).

Orienteering The Orienteering Problem (OP), which is often also referred to as the Selective Traveling Salesman Problem, originates from a sports game (Chao et al., 1996), where each player is provided with a geographical map and then has to find a tour through a subset of available checkpoints within a limited duration such that the profit collected at the checkpoints is maximized. A variant of the OP with time windows occurs as a subproblem in some of the production and distribution problems discussed in this thesis, where a subset of demanding customers and an appropriate tour through this subset must be chosen at the same time. Further generalizations of the latter problem variant include the Team Orienteering Problem with Time Windows (TOPTW), which can also be regarded as a selective VRPTW, where a fleet of vehicles needs to supply only a (best-possible) subset of demanding customers within their specified time windows.

Mathematical Programming Many articles that are referenced below present one or more mathematical programming models to formally describe the discussed optimization problems. These models firstly contain problem-specific decision variables, secondly typically linear constraints that restrict the values of the variables to certain ranges, and thirdly an objective function of the decision variables, where the function value must either be minimized or maximized subject to the given constraints. Today, there are many off-the-shelf software packages (solvers) that are capable of computing optimal solutions to such mathematical programming models, at least for small or moderate problem sizes. However, a lot of production and distribution planning problems from practice are still too complex to be solved to optimality by current standard software in a reasonable amount of time. Thus, mathematical programming models are very useful for formal definitions of such problems, but they do not necessarily imply efficient solution methods. Excellent introductions into the topic of (mixed-integer) linear programming are given by Chvatal (1983) and Vanderbei (2008).

Metaheuristics The term “meta-heuristic” was coined by Glover (1986), who describes it in the context of tabu search as an algorithm “superimposed on another heuristic. The approach undertakes to transcend local optimality by a strategy of forbidding (or, more broadly, penalizing) certain moves”. Such procedures are widely used for the solution of integrated production and distribution planning problems, as well as for complex combinatorial optimization problems in general. The ability of metaheuristics to find high-quality solutions to such problems quickly makes them preferable for many practical applications, where it is not always necessary to find an optimal solution via exact algorithms that might require large runtimes.

1.3.1 Make-to-order manufacturing models

Most of the models and algorithms presented in later chapters of this thesis are based on the work by Armstrong et al. (2008). We do not survey the contents of their paper here because a detailed discussion that forms the basis for our extensions and solution approaches is included in Chapter 2.

Chen and Vairaktarakis (2005) study an integrated production and distribution problem where orders must be delivered right after their production completion time, in an attempt to minimize finished goods inventory. The problem is drawn from applications in the computer and foot-catering industries. In their model the authors consider an unlimited number of vehicles with a fixed capacity and non-instantaneous travel times to customers. The overall objective is to maximize the customer service level, which may correspond to the minimization of the average or maximal time-span between order placement and delivery while minimizing the costs of distribution. The authors consider eight possible variations of the given problem: single vs. parallel production lines, single vs. multiple customers, and minimizing average delivery time vs. maximum delivery time. For each of these scenarios, either an efficient, polynomial-time algorithm is presented, or, in case of $\mathcal{NP}$-hardness, an appropriate heuristic method together with a performance analysis is given. Empirical results obtained on random test instances indicate that the proposed heuristics yield near-optimal solutions in reasonable running-times. The paper is one of only a few sources that include an analysis of the value of integration, i.e. a comparison of related integrated and sequential approaches for production and distribution problems. In their case, a significant improvement can be achieved by integration, mainly when average delivery times are taken into account for the service level.

The research conducted by Geismar et al. (2008) is motivated by a production and transportation scheduling problem encountered at a manufacturer of industrial adhesive chemicals, which are used for the production of plywood panels. Since the adhesiveness of the material decreases quickly, the product is considered perishable, which is directly reflected in their model. It is assumed that there is a single production facility with a constant production rate and a single, capacitated vehicle for the deliveries. The vehicle is allowed to return to the facility for reloading and to perform multiple trips during the planning period. The goal consists in determining the minimum makespan of a combined production-distribution schedule for a given set of supplied customers, subject to the given product lifespan, the vehicle capacity and non-negligible travel times between different locations.

Tarantilis and Kiranoudis (2001) deal with a fresh milk distribution problem for a dairy company in Greece. Their model is based on the vehicle routing problem (VRP)

with a fixed fleet of heterogeneous transporters. Due to the high complexity of the underlying scheduling problem and the company's requirement to solve it many times a week, the authors concentrate on an adaptation of threshold-accepting, which is a stochastic meta-heuristic method introduced by Dueck and Scheuer (1990).

While most papers dealing with integrated production and distribution scheduling concentrate on *heuristic* methods to solve the complex problems arising in this area, Tsirimpas et al. (2008) apply an *exact* dynamic programming approach to such a problem. In their setting, an optimal routing for a single vehicle with limited capacity that delivers a product from a central depot to customers according to a predefined sequence has to be determined. As a practical motivation for this problem the authors give the example of automated guided vehicle systems (AGVs), where a transport vehicle is typically guided along magnetic induction strips on the floor and carries parts to workcenters. In contrast to the model in Armstrong et al. (2008), where intermediate depot returns are not allowed and customer orders must be selected for production, the order acceptance problem is not part of this model because each customer must be served. This is a reasonable assumption in the above mentioned example of AGVs.

1.3.2 Commit-to-delivery business models

Stecke and Zhao (2007) investigate an integrated production and delivery scheduling problem that is faced by manufacturing companies employing a make-to-order approach, i.e. the production of a specific product is triggered by the arrival of an order rather than by stock-keeping considerations. Their model relies on the prerequisite that the transportation is performed by a third-party logistics provider (3PL) and that multiple shipping modes with different shipping time guarantees are available. Furthermore, fixed delivery departure dates are taken into account, that is, the 3PL provider collects finished goods e.g. once per workday. The considered business model is called *commit-to-delivery*, which means that the company chooses (and pays) the shipping mode according to the delivery time guarantee required by the customer. Stecke and Zhao model different shipping modes by introducing heterogeneous vehicles. They distinguish between splittable und unsplittable deliveries, where the splittable case can be tackled by polynomial-time algorithms for the minimum cost flow problem, while the problem with unsplittable deliveries is shown to be strongly $\mathcal{NP}$-hard. For the latter case, a polynomial-time heuristic is presented.

Zhong et al. (2010) study the same problem as Stecke and Zhao (2007) for unsplittable deliveries with a less general shipping cost structure. The goal is to determine a production schedule for accepted orders and a shipping mode for each finished

product such that the total shipping cost is minimized. An earliest-due-date-based approximation algorithm is presented with a worst-case performance ratio of 2.

Another article based on the work of Stecke and Zhao (2007) was written by Melo and Wolsey (2010). Their work focuses on closing the (small) optimality gap in the case of unsplittable deliveries between the results of the heuristic and the lower bound presented by Stecke and Zhao. To achieve this, Melo and Wolsey introduce two alternative mixed integer programming formulations for the above-named problem and present computational results.

An integrated production-distribution scheduling model in the setting of a make-to-order supply chain is presented by Chen and Pundoor (2009). They consider only one supplier and one customer, assuming that the customer places a *set* of orders, while the aim of the supplier is to find jointly a schedule for order processing on a single production line and a way of packing delivery batches such that the number of needed batches is minimized. This minimization is subject to a service-level constraint, which might consist in meeting order deadlines or the requirement to keep the average delivery lead time below a given threshold. Similar to Stecke and Zhao (2007), a 3PL-provider is responsible for the delivery of finished batches. Chen and Pundoor clarify the complexity of the considered variations of their model and develop appropriate heuristics for the $\mathcal{NP}$-hard cases. An analysis of worst-case-bounds for the presented heuristics is also included.

1.3.3 Review articles

In 1999, Sarmiento and Nagi published a review of integrated analysis of production-distribution systems. In their paper, they focus on identifying important areas where further research is needed in this setting. They state that "a unified body of literature that deals comprehensively with these types of *integrated analyses* does not exist yet". Thus, the article concentrates on showing how logistics aspects have been included in the integrated analysis and what benefits can be drawn from the integration of production and distribution planning. Since production and distribution are often linked by an intermediate inventory stage, virtually all reviewed models contain inventory decisions.

Concerning the case of perishable products, the authors refer to several models (Federgruen and Zipkin, 1984; Federgruen et al., 1986), where one-period problems for products with limited lifespan are considered. These papers include a comparison of integrated vs. sequential planning approaches, reporting substantial savings in terms of travel costs for the combined approach.

According to Sarmiento and Nagi (1999), the work conducted by Chandra and Fisher (1994) is the first that analyzes the value of integration of production scheduling and vehicle routing problems. In the latter article, a two-stage system is examined, including a single production plant with finished goods inventory in the first stage and multiple customers (retailers) in the second stage. For this setting, Chandra and Fisher report a reduction in operational costs ranging from 3–20%, when comparing sequential and integrated planning approaches. Summarizing their observations, Sarmiento and Nagi (1999) point out that "production and distribution can be completely decoupled if there is a sufficiently large inventory between them, however, the trend towards reduced inventory levels will cause the two functions [i.e., stages] to have a closer interaction".

A more recent review of integrated production and outbound distribution scheduling (IPODS) is given by Chen (2010), who conducts a general survey of the topic and includes suggestions for extensions. Chen considers scenarios, where there is typically a close link between production and distribution due to make-to-order business models (Stecke and Zhao, 2007) or time-sensitive products, such as perishable or seasonal ones. In constrast to the more strategic models surveyed by Sarmiento and Nagi (1999), Chen's review focuses on models where "there is little or no finished product inventory in the supply chain". These models belong to the operational planning level, i.e. they try to optimize detailed order-by-order production and delivery in a joint manner. Chen states that "the majority of the work in this area was done in the past five years". Furthermore, the author observes that, traditionally, many companies perform production and distribution scheduling sequentially, which he considers a suboptimal approach. Especially in case of time-sensitive products with a short lifespan, such as ready-mix concrete (Garcia and Lozano, 2004, 2005), industrial adhesive chemicals (Armstrong et al., 2008; Geismar et al., 2008) and daily newspapers (Hurter and Van Buer, 1996; Van Buer et al., 1999), production and distribution are closely coupled with almost no stock-keeping and only short time periods between the completion of processing and the delivery of orders. A further contribution by Chen (2010) consists in the proposal of a five-field notation $\alpha|\beta|\pi|\delta|\gamma$, offering a concise representation of most IPODS models in the literature. This notation is derived from the well-known three-field notation $\alpha|\beta|\gamma$ as introduced by Graham et al. (1979) for production scheduling models. In the five-field notation, α represents the machine-configuration at the production plant(s), β describes constraints concerning the orders (e.g. release dates or customer time windows), π specifies characteristics of the delivery vehicle(s) (e.g. their number and capacity), δ determines the number of customers and γ finally describes the associated objective function, which is typically a combination of customer service level, transportation costs, and revenues.

1.3.4 Applications of (meta-) heuristics

Lei et al. (2007) investigate a multi-period production and distribution process involving a heterogeneous fleet of vehicles. Their application is drawn from the petrochemical industry. For the most general variant of the problem under consideration an $\mathcal{NP}$-hardness result is obtained, whereas some special cases, where the problem can be formulated as a particular minimum cost flow problem, are shown to be polynomially solvable. As a solution method, the authors present a linear programming relaxation based heuristic, which is reported to yield error gaps of about 1% (within 0.25s) compared to the best feasible solutions obtained by running CPLEX 9.1 with a time limit of one hour per instance on the considered set of test instances.

Boudia and Prins (2009) deal with the minimization of production setups, inventory and distribution costs in a multi-period production and distribution problem, where a set of identical vehicles are responsible for the deliveries of a single plant with a limited production capacity. Storage space is available at the plant as well as at the customer sites. The deliveries are scheduled in a vendor-managed inventory replenishment (VMI) business mode, where customers do not place specific orders, but the delivery quantities are determined as an integral part of the planning process. The authors solve the problem by employing a memetic algorithm (a genetic algorithm combined with local search as a mutation step) that is equipped with a method for population management. A comparison with an existing *greedy randomized adaptive search procedure (GRASP)* devised by the same authors shows that the memetic approach is superior.

Further research in the scenario of vendor-managed inventory replenishment is conducted by Bard and Nananukul (2009), where multiple vehicles serve a set of customers with time-varying demands from a single facility. The deliveries are used to restock the customers' inventory or to satisfy their daily demands. This integrated production-inventory-routing problem is tackled by a metaheuristic approach called *reactive tabu search with path relinking* (Glover and Laguna, 1997; Resende and Ribeiro, 2005). Extensive computational results on the instances from Boudia and Prins (2009) are presented, including comparisons with the results of their GRASP procedure and a MIP formulation solved (for smaller instances) via CPLEX. The reactive tabu search proved very effective for the considered test cases, whereas the authors admit that the improvements come at a three- to five-fold increase in runtime compared to the results of the GRASP algorithm.

Cornillier et al. (2009) study a petrol station replenishment problem with time windows faced by a transportation company in Quebec. In this problem, the transportation

of several petroleum products from a central depot with a heterogeneous fleet of vehicles (with different compartments for products) must be planned for a working day, including decisions about delivery quantities, compartment assignments, and vehicle routing. The aim is to maximize the sales revenue, minus traveling costs and regular and overtime costs. The authors present a MIP formulation for the given problem and propose two heuristics: the first employs a strategy called arc preselection, which is based on nearest-neighbor and minimum spanning tree considerations. The second heuristic (intended for larger instances) uses a decomposition approach to divide the geographically dispersed customers into sectors, where the problem of each sector is solved independently using a branch and bound algorithm. Computational experiments have shown that the proposed heuristics are capable of finding near-optimal solutions in a small amount of time.

1.3.5 The (Team) Orienteering Problem

A discussion of so-called Traveling Salesman Problems (TSPs) with profits is presented by Feillet et al. (2005). In this generalization of the classical TSP, profit values are associated with each node of the transport graph and it is not required that each vertex is visited. The objective consists in the simultaneous optimization of collected profit and resulting travel costs. The authors classify TSPs with profits by defining three generic variants of the problem: (1) collected profit and travel costs both appear in the objective function, (2) collected profit is maximized subject to a predefined upper bound on the travel costs, (3) travel costs are minimized subject to a predefined lower bound on the collected profit.

Th most interesting variant for our considerations is generic problem (2), which is often encountered in the literature as the *orienteering problem* (*OP*, Tsiligirides, 1984; Ramesh et al., 1992; Vansteenwegen et al., 2011), or the *selective TSP* (Laporte and Martello, 1990; Millar and Kiragu, 1997; Gendreau et al., 1998), which is of particular importance as a subproblem in many of the integrated production and distribution planning problems mentioned in this thesis. This is due to the fact that the decision to supply a customer can be regarded as collecting a profit at the correspoding node, while limited product lifespans place a natural restriction on the length of a feasible distribution tour.

Archetti et al. (2007) extend the work of Feillet et al. (2005) by investigating vehicle routing problems (VRPs) with profits, i.e. a generalization of TSPs with profits to the multi-vehicle case, or, more precisely, a generalization of the OP, called the *team orienteering problem (TOP)*. The TOP was first introduced by Chao et al. (1996) and

continues to be an active area of research (Tang and Miller-Hooks, 2005; Boussier et al., 2007; Souffriau et al., 2010).

A variant of the (single-vehicle) orienteering problem that respects time window constraints at customers, called the OPTW, has been studied by Kantor and Rosenwein (1992), where extensions of heuristics for the OP are developed to solve the OPTW. More recent research (Vansteenwegen et al., 2009a,b) considers the TOP with time windows (TOPTW) in the setting of personalized electronic tourist guides, where the TOPTW is solved heuristically by an iterated local search approach. An exact optimization algorithm for the OPTW is presented in Righini and Salani (2006, 2009), where the authors employ the dynamic programming paradigm in combination with a technique called "decremental state space relaxation" that serves as a means of bounding the exponentially growing state space. An iterated local search approach for the TOPTW is discussed in the PhD thesis of Souffriau (2010). Recently, the same problem has also been addressed via ant-colony optimization in Montemanni and Gambardella (2009).

Chapter 2

The basic integrated production and distribution model

The models and algorithms discussed in this dissertation are mainly inspired by a basic integrated production and distribution model that has been introduced by Armstrong et al. (2008). In this chapter, we present a detailed description of this model and establish a common notation which is used throughout the thesis. Section 2.1 primarily contains this formulation, together with a corresponding mixed integer linear program and a view on the complexity aspects of the problem. Furthermore, Section 2.2 describes the existing branch and bound approach, which in turn employs a greedy heuristic presented in Section 2.3.

2.1 Model formulation

Armstrong et al. (2008) introduced an **integrated production and distribution problem**, where at a production facility (or depot) a single product with a **limited lifespan** B is produced. The product lifespan (also called shelf-life) specifies that the product expires B time units after the end of its production. In the model, the production facility consists of a single machine, which is capable of processing at most one order at any time and the production is carried out with a **constant production rate** R, meaning that R units of the product can be produced in each time unit.

There are n potential **customers** $i = 1, \ldots, n$ (additionally, the production facility is denoted by 0) and each customer i places a single order to satisfy a **demand** of p_i units of the product. Furthermore, each customer specifies a **time window** $[a_i, b_i]$,

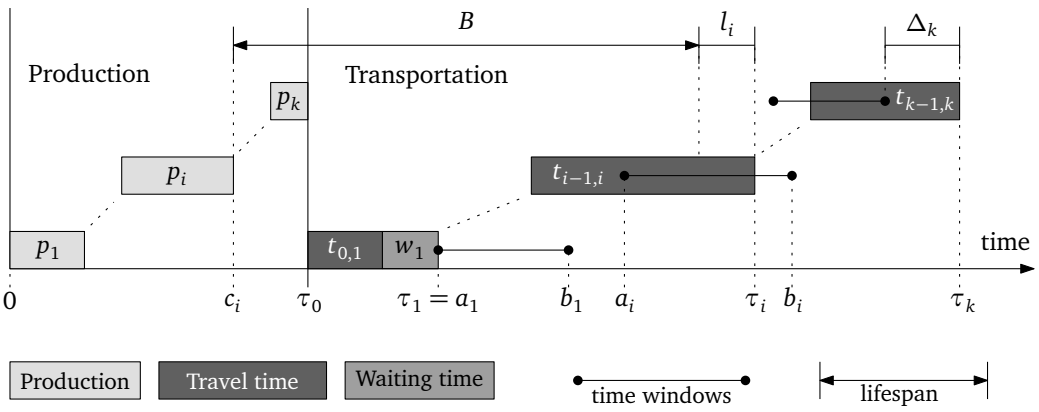


Figure 2.1: Times and periods in the production and transportation stages

within which the order should be delivered. Orders are transported from the facility to the customers by a **single vehicle** that has sufficient capacity to deliver all possible orders on a single tour. The transportation starts when all accepted orders have been produced. The **travel times** t_{ij} between locations i and j (t_{0i} between the facility 0 and customer i) are assumed to be non-negligible, known in advance, and to obey the triangle inequality, which means that traveling from location i to j takes no longer than traveling from i to j via some other location k, i.e. $t_{ij} \leq t_{ik} + t_{kj}$ for all i, j, k. If the vehicle arrives at customer i before the beginning a_i of the delivery time window, a **waiting time** w_i is incurred and the goods are not delivered until time a_i (see for example order 1 in Figure 2.1). In contrast to this, late deliveries that exceed the deadline b_i for a customer i are not allowed. This is motivated by the fact that in some settings late goods are assumed to cause additional costs, e.g. for an environmentally-safe disposal of harmful substances.

According to certain restrictions in practice, Armstrong et al. assume that the sequences for production and transportation of the goods are the same and fixed. Thus, the goods are produced in the same order as they are delivered. We assume without loss of generality that the given sequence is $S = (1, \ldots, n)$. Since the facility has a limited production rate and the transportation by the single vehicle is subject to lifespan and time window constraints, it is likely that not all customers can be supplied in an admissible way. The problem now consists in choosing a subsequence $\sigma \subseteq S$ of customers that satisfies the following properties.

(1) Each customer $i \in \sigma$ is supplied within the specified time window $[a_i, b_i]$.

(2) Each order is delivered at most B time units after its production completion time.

(3) The total amount of demand satisfied by the deliveries is maximal.

In the following we will denote by τ_i the **delivery time** at customer i, which has to be determined for every customer included in the chosen subsequence. The production and transportation phases are illustrated in Figure 2.1: on the horizontal axis, discrete time points are given as defined above. In the period from time 0 to τ_0, some orders between 1 and k are produced consecutively (without idle times) with production times proportional to the individual demand of the product. Each customer is assigned a "row" in the diagram. The production time p_i for order i ends at time c_i, which is also the beginning of the associated lifespan. The latter lasts from c_i to $c_i + B$, marked by the duration B in the figure.

At time τ_0 all selected orders have been produced. Immediately after that, the transportation phase (also called distribution phase) begins. Since the customers are visited in the same sequence as the corresponding orders are produced, the vehicle leaves the depot and travels to the first selected customer. As illustrated by the period w_1 in the diagram, there may be waiting times for the vehicle, if it arrives early. Then, the delivery time at the customer equals the beginning of the associated delivery time window, which is symbolized in each row of the diagram by the lines with dots at both ends. A production and distribution schedule is called **feasible**, if the delivery time τ_i is between a_i and b_i for each supplied customer i and additionally no customer receives expired products (i.e. $\tau_i \leq c_i + B$ for each order $i \in \sigma$).

Sometimes we also consider **infeasible schedules**, where the customer time windows or the lifespan of the product may be exceeded by the deliveries. Missing a deadline b_i causes a positive **delivery tardiness** $\Delta_i = \max\{0, \tau_i - b_i\}$ in the schedule (order k in Figure 2.1 is an example), while delivering an order after the end of its lifespan results in a positive **lifespan tardiness** $l_i = \max\{0, \tau_i - (c_i + B)\}$ (see order i in Figure 2.1).

Many models in transportation scheduling explicitly consider **service times** at customers (see e.g. Azi et al. (2007); Chen (2010)) which may, for example, represent the duration of vehicle unloading processes. However, these service times are not regarded in this model because they can be incorporated into the travel times, if necessary. For this reason, there is practically no distinction between delivery times and departure times at customers.

As the production rate R of the facility is constant, we can assume w.l.o.g. that $R = 1$ because all other time-dependent parameters (travel times, product lifespan and time windows) can be scaled (multiplied by R) for subsequent calculations and scaled back (divided by R) to obtain the actual solution. For this reason we can write p_i for both the demand and the duration of the production associated with order i. This will be useful in the presentation of mathematical models later on.

In the model, a finite planning period is considered. This motivates the assumption that each customer is only allowed to place a single order. Thus, in our further description we will use the notions of a customer and the associated order interchangeably.

2.1.1 A mixed-integer linear programming model

Armstrong et al. present a mixed integer linear programming (MIP) model, which is solved for a comparison between their own exact branch and bound algorithm and the solution of the MIP-model via the commercial solver IBM ILOG CPLEX[1]. The MIP-model is given by the following formulation which is based on binary network flow variables $Y_{i,j}$ (2.1g), indicating whether customer j is visited directly after customer i ($Y_{i,j} = 1$) or not ($Y_{i,j} = 0$). For this purpose, two nodes, 0 and $n+1$, are introduced to represent the production facility, where we distinguish between departure 0 and arrival $n+1$ of the planned tour. Furthermore, variables w_i (2.1h) stand for the waiting time for the vehicle at customer i.

Using this notation, we can calculate the delivery time τ_k at customer k by the following summation:

$$\tau_k := \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} p_j Y_{i,j} + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k} t_{i,j} Y_{i,j} + \sum_{i=1}^{k} w_i$$

The value of τ_k is only meaningful if k is actually selected in a solution, i.e. only when the vehicle travels from another node to k. This, in turn, is true if $\sum_{i=0}^{k-1} Y_{i,k} = 1$. The definition is composed of three terms: the first is the total production time for all produced orders, the second represents the total travel time from the production facility to customer k and the third accounts for all waiting times before the delivery at k. Moreover, in order to formulate lifespan constraints we need to calculate the production finishing times for all orders k, which we will denote by c_k as follows:

$$c_k := \sum_{i=0}^{n-1} \sum_{j=i+1}^{k} p_j Y_{i,j}$$

We will now use these definitions as abbreviations to compactly state the following mixed integer program.

[1] http://www.ibm.com/software/integration/optimization/cplex

$$\text{Maximize} \quad P = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} p_j Y_{i,j} \tag{2.1a}$$

subject to

The time window constraints:

$$\tau_k \geq a_k \sum_{i=0}^{k-1} Y_{i,k} \qquad (k = 1, \ldots, n) \tag{2.1b}$$

$$\tau_k \leq b_k + M \left(1 - \sum_{i=0}^{k-1} Y_{i,k} \right) \qquad (k = 1, \ldots, n) \tag{2.1c}$$

The lifespan constraints:

$$\tau_k - c_k \leq B \qquad (k = 1, \ldots, n) \tag{2.1d}$$

The flow conservation constraints (using a dummy node $n+1$):

$$\sum_{h=0}^{i-1} Y_{h,i} = \sum_{j=i+1}^{n+1} Y_{i,j} \qquad (i = 1 \ldots, n) \tag{2.1e}$$

$$\sum_{h=1}^{n+1} Y_{0,h} = 1 \tag{2.1f}$$

Using the variables:

$$Y_{i,j} \in \{0, 1\} \qquad (0 \leq i < j \leq n+1) \tag{2.1g}$$

$$w_i \geq 0 \qquad (i = 1, \ldots, n) \tag{2.1h}$$

The objective, given in equation (2.1a), is to maximize the sum of all demands of customers served. Note that in this model a customer $j \in \{1, \ldots, n\}$ is visited, if and only if $\sum_{i=0}^{n-1} Y_{i,j} = 1$. Inequalities (2.1b) and (2.1c) represent the time window constraints. Stating restriction (2.1c) requires a "big-M" formulation (with a sufficiently large constant M) that essentially enables the restriction, if customer k is served and disables it, otherwise. Furthermore, the lifespan constraints are given by inequalities (2.1d), ensuring that for each customer k the time between the delivery at k and the

completion of the corresponding order is at most the product lifespan B. Finally, the flow conservation constraints (2.1e) and (2.1f) require the vehicle to leave the depot node 0, visit the chosen customers and finish the tour at node $n+1$, which again denotes the depot.

2.1.2 Complexity

The integrated production and distribution problem introduced in Section 2.1 can easily be proved to be $\mathcal{NP}$-hard in the ordinary sense because it contains the $\mathcal{NP}$-hard binary knapsack problem (Garey and Johnson, 1979, MP9) as a special case. This can be seen by considering a variant of the problem, where the time window constraints are relaxed ($a_i = 0$ and $b_i = \infty$ for all customers i), instantaneous travel times are assumed ($t_{i,j} = 0$ for all locations i, j) and it is required that $p_1 > \sum_{j=2,\dots,n} p_j$ such that customer 1 is always selected in an optimal solution. Applying these restrictions, the remaining problem can be formulated as follows:

$$\begin{aligned} \text{Maximize } P_{\text{relaxed}} = & \sum_{j=2}^{n} p_j x_j \\ \text{s.t.} \quad & \sum_{j=2}^{n} p_j x_j \leq B \\ & x_j \in \{0,1\} \qquad (j = 2, \dots, n) \end{aligned} \tag{2.2}$$

This integer programming formulation contains binary variables x_j for each customer $j = 2, \dots, n$ (other than the first customer) which indicate whether customer j is selected or not. Customers together with their demands can be regarded as the "knapsack items", whereas the capacity of the knapsack is given by the lifespan B, i.e. the duration following the production of order 1 that is available to produce a subset of the orders $j = 2, \dots, n$.

As an alternative to solving the problem at hand using MIP techniques, Wendt (2009) describes a dynamic programming algorithm that has a pseudopolynomial running time which depends on the number of customers, a predefined finite time horizon, the total demand ordered and the maximum width of time windows.

In order to cope with the apparent intractability of this problem, Armstrong et al. propose an exact branch and bound algorithm which employs a heuristic method (see Section 2.3) to obtain lower bounds for the optimal solution value. In the following subsection we describe the main aspects of this branch and bound method, including a discussion of a mistake in the handling of waiting times for the computation of bounds, which we will correct later on.

2.2 Branch and bound for the basic model

The branch and bound algorithm described by Armstrong et al. constructs a search-tree with (at most) 2^n nodes, representing all possible subsets of n customers. An example of such a tree for $n = 5$ customers is shown in Figure 2.2, where each path from the root node 0 to another node in the tree corresponds to a possible selection of customers. In the example, the path indicated by the shaded nodes represents the selection $(2, 4)$. As such paths are unique for a given node in the tree, we can also identify selections with nodes rather than with paths. In this case, shaded node "4" is the representative for the selection. Note that in this representation *also inner nodes* of the tree are associated with customer selections (potential solutions), and not only leaf nodes.

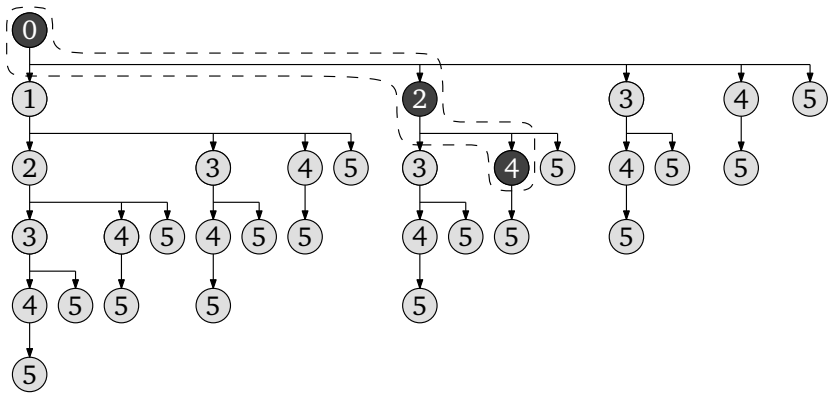


Figure 2.2: A complete branch and bound-tree for $n = 5$ customers with an example customer selection $\sigma = (2, 4)$ as indicated by the shaded nodes

Let $\sigma = (\sigma_1, \ldots, \sigma_k) \subseteq S$ denote the current customer selection corresponding to a node v_σ in the search tree. Then, for the branching step, all possible customers i with $\sigma_k < i \leq n$ are considered in turn for the construction of children $v_{\sigma^{(i)}}$ of v_σ. These children belong to the sequences $\sigma^{(i)} := \sigma \parallel i = (\sigma_1, \ldots, \sigma_k, i)$, where $i = \sigma_k + 1, \ldots, n$. Thus, the current sequence σ is extended by each of the possible children i in turn (see Figure 2.3). Clearly, all nodes associated with sequences that include the last customer n are leaf nodes of the tree because for these nodes there is no customer left to append.

2.2.1 Feasibility of branching steps

In the following we will give a description of the criteria used by Armstrong et al. to determine the feasibility of a branching step and we will discuss the bounding scheme

used in their paper. Unfortunately, the branching as well as the bounding are flawed, but these steps can be corrected, as we will show afterwards. For this reason we will first present the original methods and provide the corrections after that.

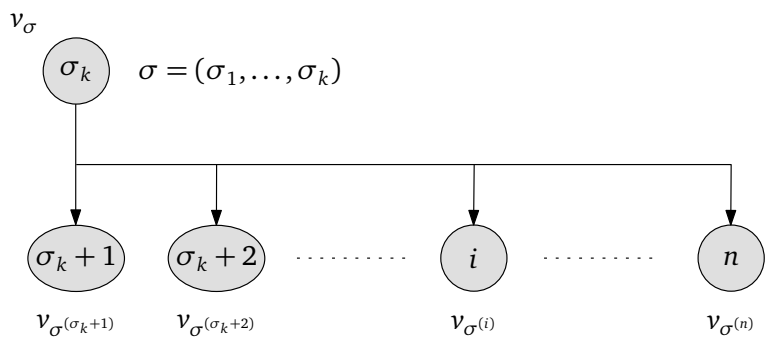


Figure 2.3: Branching in the search tree

For an order/customer $j \in \sigma$, recall that w_j denotes the associated waiting time, τ_j is the delivery time and c_j the production completion time. The following values are used in Armstrong et al. (2008) to derive criteria for the feasibility of a branching step:

- $s_\sigma^T = \min_{j \in \sigma} \left\{ w_j + (b_j - \tau_j) \right\}$ is the maximum delay in the delivery times (over all orders $\sigma_1, \ldots, \sigma_k$) without making any scheduled order in σ tardy.
- $s_\sigma^L = \min_{j \in \sigma} \left\{ B - (\tau_j - c_j) \right\}$ is the maximum delay in the delivery times without violating the lifespan constraint for any scheduled order in σ.
- For each $i \notin \sigma$ the demand p_i corresponds to the delay of the transportation start caused by the additional production time for order i, when appended.

It is a simple task to calculate the values s_σ^T and s_σ^L whenever the search explores a node v_σ in the search tree. The authors propose to check the following two conditions when evaluating a branching step from v_σ to $v_{\sigma^{(i)}}$, where customer i is appended to selection σ (see Figure 2.3). If either condition is satisfied, the considered branch should be cut off.

$$\tau_i > b_i \tag{2.3}$$

$$p_i > \min \left\{ s_\sigma^T, s_\sigma^L \right\} \tag{2.4}$$

If in inequality (2.3) the delivery time τ_i for the appended customer i exceeds the deadline b_i, then no feasible schedule can be obtained in the subtree rooted at $v_{\sigma^{(i)}}$.

The reason for this is that by adding further customers the delivery tardiness at i cannot be removed.

The second inequality (2.4) given in their paper contains an incorrect right-hand side, which we will demonstrate in Example 1 in the next chapter (p. 28). The general idea is to cut off a branch if the additional production time p_i is larger than the allowed delay of the transportation start. We will present a corrected right-hand side for this condition later on.

2.2.2 Upper bounds

At each (inner) node of the search tree we can calculate an upper bound G^*_σ for the maximum gain in satisfied demand, based on the current customer selection $\sigma = (\sigma_1, \ldots, \sigma_k)$, by relaxing the time-related constraints to some extent. In other words, we assume that σ represents a partial customer selection, whose objective function value is known at the current state of the search, and our goal is to estimate the demand that can additionally be satisfied by deliveries to customers from the set $\{\sigma_k + 1, \ldots, n\}$. Then, the value of G^*_σ can be calculated by solving the following binary knapsack problem in which binary variables x_j for $j = \sigma_k + 1, \ldots, n$ are used to determine whether order j is included in the solution ($x_j = 1$) or not ($x_j = 0$).

$$\begin{aligned} \text{Maximize } G_\sigma = & \sum_{j=\sigma_k+1}^{n} p_j x_j \\ \text{s.t.} \quad & \sum_{j=\sigma_k+1}^{n} p_j x_j \leq \delta_{max} \\ & x_j \in \{0,1\} \qquad (j = \sigma_k + 1, \ldots, n) \end{aligned} \tag{2.5}$$

In this IP-formulation, the "capacity of the knapsack" is given by δ_{max}, which denotes the maximum delay for the start of transportation such that the lifespan and time window restrictions of the already included orders are still obeyed. We will describe the computation of δ_{max} later on in this section. The delay can be used to produce further orders to be included in the deliveries.

By computing the optimal solution value G^*_σ, we select a subset of currently unselected orders in such a way that the amount of goods (order quantity) which can be produced within a time period of length δ_{max} is maximized. Note that this is actually a relaxation because this computation neglects time windows as well as lifespan constraints and travel times for the hypothetically added orders. Thus, we get an upper bound on the maximum additionally satisfiable demand.

For the discussion of the bounding scheme let P^0 be any lower bound for the optimal solution value of the original problem. For instance, P^0 may have been derived by a heuristic method or found while visiting feasible solutions earlier in the search. Moreover, let σ be the current customer selection and let G^*_σ be calculated as described above. If we denote the sum of demands of all orders in σ by $p_\sigma := \sum_{j\in\sigma} p_j$, then the search tree can be pruned below a node v_σ, if

$$p_\sigma + G^*_\sigma \leq P^0 \tag{2.6}$$

because this condition implies that no descendant node of v_σ can have a larger solution value than the current best known solution value P^0. Checking whether inequality (2.6) holds requires the computation of an appropriate value for δ_{max} at each node because this value determines the current size of the knapsack. Unfortunately, this step is described incorrectly in the original paper by Armstrong et al. (2008). They use the value $\delta_{max} = \min\left\{s^T_\sigma, s^L_\sigma\right\}$, which also occurs in condition (2.4) for testing the feasibility of a branching step, as the maximum delay in the above IP-formulation (2.5) to compute the maximum gain in satisfied demand. However, the counterexample in Section 3.1 proves that this is not correct in general.

2.3 A greedy heuristic to obtain lower bounds

In order to achieve a better pruning of the branch and bound search-tree, it is important to calculate good feasible solutions early in the search process, because bounds like the one presented in equation (2.6) depend on the value of the current best known solution P^0. Since we consider a maximization problem, the value of each feasible solution yields a lower bound on the optimal solution value P^*. The generation of good feasible solutions is commonly done using efficient heuristics, like the deletion-based greedy heuristic proposed by Armstrong et al. (2008).

For a given problem instance, let $\sigma^* \subseteq S$ represent an optimal solution, i.e. a customer selection that yields maximum satisfied demand. The central idea of the deletion-based heuristic is to search for an optimal set $\varphi^* := S \setminus \sigma^*$ of customers to be excluded from the complete set S. This is accomplished via an iterative approach that starts with the complete selection $\sigma := S$ as the current solution. Since this solution will be infeasible for most non-trivial problem instances, the heuristic proceeds by greedily searching for a pair (i^*, j^*) of customers in σ, which should be permanently deleted from σ. Note that $i^* = j^*$ is possible, so during each iteration exactly one or two customers are removed from σ.

The actual choice of customers to be removed in an iteration is made as follows. For any selection $\sigma \subseteq S$ let $F(\sigma) := \{i \in \sigma \mid \Delta_i = 0,\ l_i = 0\}$ be the set of customers that are feasibly supplied (without delivery or lifespan tardiness). Furthermore, let $p(\sigma) := \sum_{i \in F(\sigma)} p_i$ denote the feasibly satisfied demand of customer selection σ. Then, in each iteration the deletion-based heuristic chooses a pair of customers (i^*, j^*), such that

$$p\left(\sigma \setminus \{i^*, j^*\}\right) = \max_{i,j \in \sigma} \left\{ p\left(\sigma \setminus \{i, j\}\right) \right\}$$

The search is continued until a feasible customer selection is obtained. The running time of the heuristic can be shown to be in $\mathcal{O}(n^4)$, where $n = |S|$ represents the total number of customers.

Chapter 3

Extensions of the basic model

The basic model of Armstrong et al. (2008) presented in Chapter 2 is rather restricted, as it is unlikely to occur in practical applications in its pure form. This is due to the many simplifying assumptions being made, such as pre-defined customer sequences, a fixed production start time, a vehicle with unlimited capacity etc. However, the basic model represents a subproblem encountered in integrated production and distribution scheduling problems and it provides a good starting point for model extensions that aim at incorporating more realistic restrictions from practice.

In this chapter, we present some corrections and additions to the basic model that we implemented and which also form a basis for further considerations in Chapters 4 and 5, where more thorough modifications like variable customer sequences and multi-vehicle models are discussed.

In Section 3.1 we point out a mistake in the calculation of upper bounds in the branch and bound algorithm of Armstrong et al. and propose a corrected variant. This computation involves solving a binary knapsack problem, which we address by dynamic programming in Section 3.2. In Section 3.3 we argue that for many problem instances better schedules can be obtained if it is allowed to delay the production start. We provide a systematic way of dealing with these delays and introduce the notion of shift-feasibility. Furthermore, in Section 3.4 we describe a tabu search metaheuristic that is designed to cope with production delays and which represents a fast and efficient alternative to the (corrected) branch and bound algorithm. Finally, Section 3.5 discusses further objective functions as alternatives to the maximization of the total satisfied demand.

3.1 Correcting the calculation of upper bounds

As described in Section 2.2, the calculation of upper bounds in the branch and bound algorithm of Armstrong et al. (2008) contains a flaw due to an incorrect handling of waiting times. In this section, we give a counterexample and provide a corrected version of that calculation.

We consider a problem instance used in the original paper mentioned above (see *Example 1*) and modify it only slightly by dividing the demands by the given production rate $R = 10$, so we can neglect R in the following calculations and the (numerical values of) demands and production times are equal.

(a) Travel times

t_{ij}	1	2	3	4	5
0	4	5	3	4	3
1		6	4	5	4
2			3	6	2
3				4	5
4					3

(b) Demands and time windows

Customer j	1	2	3	4	5
p_j	3	4	6	3	4
a_j	19	26	31	32	36
b_j	24	32	36	35	37

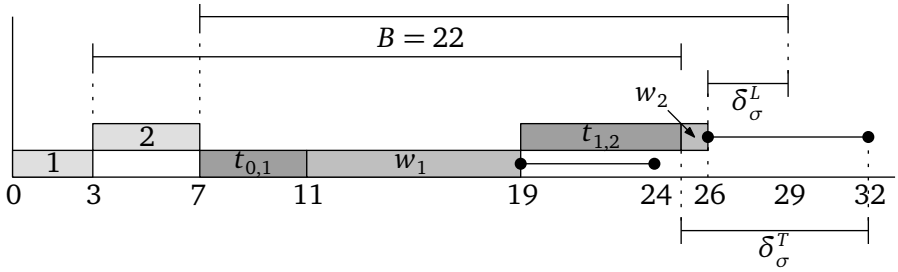
Table 3.1: Parameters for Example 1

Example 1 Consider the problem instance specified in Table 3.1, together with a lifespan of $B = 22$ time units. The proposed branch and bound algorithm adds customer 1 in the first step and customer 2 in the second. Then $\sigma = (1,2)$ is a feasible selection with a satisfied demand of 7 units. The corresponding production and transportation schedule is shown in Figure 3.1(a). In order to obtain the upper bound proposed by Armstrong et al. (2008), we need to compute the so-called *delivery-time slack* and the *lifespan-slack* of selection σ:

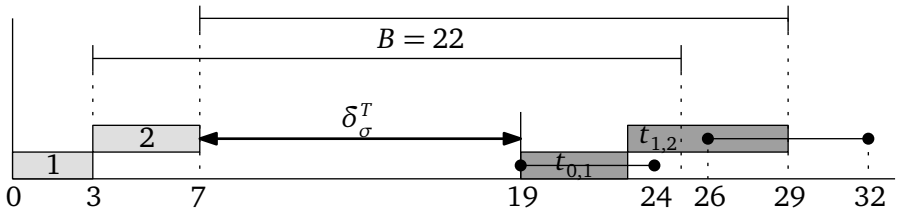
$$s_\sigma^T = \min\{24 - 19 + 8,\ 32 - 26 + 1\} = 7\,,$$
$$s_\sigma^L = \min\{22 - (19 - 3),\ 22 - (26 - 7)\} = 3\,,$$

and $\delta_{max} = \min\left\{s_\sigma^T, s_\sigma^L\right\} = 3$. This value now negatively affects the branching as well as the bounding calculations:

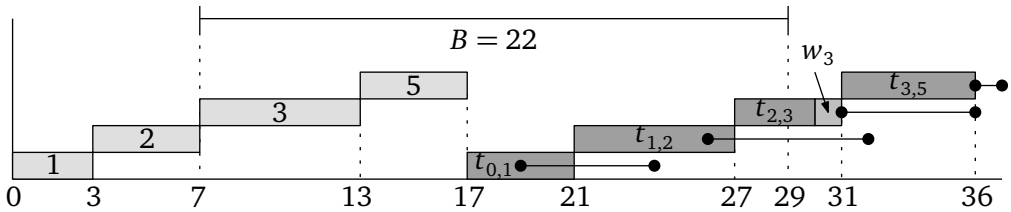
Branching The branch to append customer 3 will be cut off, because $p_3 = 6 > 3 = \delta_{max}$, satisfying condition (2.4). This means that all selections beginning with $1, 2, 3, \ldots$ are not considered later on. However, as Armstrong et al. show, the selection $\sigma^* = (1,2,3,5)$ in Figure 3.1(c) is the unique optimal solution with a satisfied demand of 17 units.



(a) Production and transportation schedule for selection $\sigma = (1,2)$



(b) The maximum transport delay for $\sigma = (1,2)$ is given by $\delta_\sigma^T = 12$



(c) Optimal schedule for selection $\sigma^* = (1,2,3,5)$

Figure 3.1: Schedules for Example 1

Bounding Solving IP (2.5) for $\sigma = (1,2)$ and $\delta_{max} = 3$ yields the solution $x_4 = 1$, $x_3 = x_5 = 0$. Thus, the upper bound at this node in the search tree is $\sum_{i \in \sigma} p_i + p_4 = 10$. This is clearly wrong because $\sigma^* = (1,2,3,5)$ is a descendant of σ, admits a feasible schedule and has a larger objective value of 17.

Thus, using the values s_σ^T and s_σ^L yields incorrect branching and bounding steps in this situation. ◊

The mistake in this calculation consists in the fact that waiting times at customers are not handled properly by defining $\delta_{max} = \min\left\{s_\sigma^T, s_\sigma^L\right\}$, since additional time spent for production (making the vehicle wait at the production facility) does *not* necessarily

delay the delivery time at any customer in the sequence due to waiting times. These may allow to delay the start of the transportation stage even further than the value $\min\left\{s_\sigma^T, s_\sigma^L\right\}$ indicates. Thus, using the previously defined value for δ_{max} does not always yield a true upper bound for the delay of the transportation start.

Nevertheless, we can correct the upper bound calculation by defining δ_{max} as the maximum transport delay δ_σ^T of sequence σ, which can be computed as becomes obvious from the following definition.

Definition 1 *Let $\sigma = (\sigma_1, \ldots, \sigma_k)$ be a selection of orders that admits a feasible schedule, characterized by production finishing times c_{σ_i}, waiting times w_{σ_i}, and delivery times τ_{σ_i} for each $\sigma_i \in \sigma$. The* ***maximum transport delay*** *$\delta_{\sigma_i}^T$ of an order $\sigma_i \in \sigma$ is recursively defined by*

$$\delta_{\sigma_i}^T := w_{\sigma_i} + \min\left\{b_{\sigma_i} - \tau_{\sigma_i},\ c_{\sigma_i} + B - \tau_{\sigma_i},\ \delta_{\sigma_{i+1}}^T\right\} \quad \text{for } i = k, \ldots, 1, \tag{3.1}$$

where initially $\delta_{\sigma_{k+1}}^T := \infty$ for a dummy order σ_{k+1}. The ***maximum transport delay of the selection σ*** *is defined as the maximum transport delay of the first order in σ: $\delta_\sigma^T := \delta_{\sigma_1}^T$.*

This definition assumes that a feasible schedule for selection σ is given by the variables c_{σ_i}, w_{σ_i}, and τ_{σ_i}, $\sigma_i \in \sigma$. Based on that, the maximum transport delay $\delta_{\sigma_i}^T$ of an order $\sigma_i \in \sigma$ is composed of the waiting time w_{σ_i} and an additional delay which is determined by the minimum of three values: the slack in the delivery time window, $b_{\sigma_i} - \tau_{\sigma_i}$, the slack in the corresponding lifespan, $c_{\sigma_i} + B - \tau_{\sigma_i}$, and the possible delay permitted by the subsequent orders, $\delta_{\sigma_{i+1}}^T$.

To illustrate Def. 1, let us again consider Example 1 together with the associated schedule for selection $\sigma = (1, 2)$, shown in Figure 3.1(a). We calculate the values

$$\delta_2^T = w_2 + \min\{32 - 26,\ 7 + 22 - 26,\ \infty\} = 1 + 3 = 4,$$
$$\delta_1^T = w_1 + \min\{24 - 19,\ 3 + 22 - 19,\ 4\} = 8 + 4 = 12,$$

and obtain the maximum transport delay $\delta_\sigma^T = \delta_1^T = 12$. Shifting the transport start by δ_σ^T yields the schedule shown in Figure 3.1(b). Since it would take 13 time units to produce the remaining orders 3, 4 and 5, we can also derive that it is impossible to supply all customers during the current planning period (on a single trip).

The maximum transport delay δ_σ^T is also important for deciding whether a branching step is feasible. Due to the above considerations, inequality (2.4) needs to be changed to

$$p_i > \delta_\sigma^T \tag{2.4'}$$

for correct branchings. A further necessary condition for feasibility which was not used by Armstrong et al. (2008) concerns the lifespan constraint associated with the appended customer i in the branching step:

$$\tau_i \leq \tau_0 + B. \tag{3.2}$$

Condition (3.2) states that the delivery time at the appended customer i must not exceed the expiry time $\tau_0 + B$ of order i. Note that $\tau_0 = c_i$, i.e. the start of transportation is the production finishing time of the appended order i. If this condition is violated, order i cannot be delivered within the product lifespan for the current customer selection. In this case the branching step is infeasible and the considered branch should be cut off.

Setting $\delta_{max} = \delta_\sigma^T$ in IP-formulation (2.5) yields correct upper bounds, since delaying the start of transportation by δ_σ^T is feasible w.r.t. time windows and lifespans, whereas a delay beyond δ_σ^T time units will make the schedule become infeasible by definition.

3.2 Computing maximum gains by dynamic programming

Although computing the maximum gain in satisfied demand, G_σ^*, by solving the binary knapsack problem (2.5) is $\mathcal{NP}$-hard in the ordinary sense (see Garey and Johnson, 1979, MP9), solutions for quite large instances can be calculated by the well-known dynamic programming algorithm for this problem (see e.g. Papadimitriou and Steiglitz, 1982, ch. 17.3). Since the problem sizes that are tractable for this algorithm exceed the assumed sizes for the original production and distribution scheduling problem, such an approach is suitable for computing G_σ^* as an alternative to solving IP-model (2.5) directly via standard MIP-solvers.

Armstrong et al. (2008) do not mention how they solve the problem of computing G_σ^*. Here, we employ an own adaptation of the dynamic programming method for this problem, which is shown in Algorithm 3.1. It consists of a procedure ComputeMaxGainValues(S), which is called exactly once in the solution process, namely before the execution of the branch and bound algorithm.

The procedure first computes the value $W = \max_{i \in S} \{b_i - (p_i + t_{0i})\}$ in step 3. This is an upper bound on the possible additional production time if at least one of the

Algorithm 3.1 Computing maximum gain values by dynamic programming

1: **procedure** COMPUTEMAXGAINVALUES(S)
2: $n \leftarrow |S|$
3: $W \leftarrow \max_{i \in S} \{b_i - (p_i + t_{0i})\}$
4: Initialize $G(n+1, w) \leftarrow 0$ for all $w \in \{0, \ldots, W\}$
5: **for** $j \leftarrow n, n-1, \ldots, 2, 1$ **do**
6: **for** $w \leftarrow 0, 1, \ldots, W-1, W$ **do**
7: **if** $w < p_j$ **or** $G(j+1, w-p_j) + p_j \leq G(j+1, w)$ **then**
8: $G(j, w) \leftarrow G(j+1, w)$
9: **else**
10: $G(j, w) \leftarrow G(j+1, w-p_j) + p_j$

orders in S has already been scheduled for delivery. Under the assumption that all time-related parameters have been scaled for a production rate of $R = 1$, we can use W as a "largest knapsack size" in the algorithm. After that, variables $G(j, w)$ are defined for $j = 1, \ldots, n+1$ and $w = 0, \ldots, W$, where $G(j, w)$ denotes the maximum possible gain in satisfied demand, when only customers $j, \ldots, n$ (a suffix of the sequence S) and a delay of the transportation start of w time units are considered. These values are iteratively computed using the following recursion for all $w \in \{0, \ldots, W\}$:

$$G(j, w) = \begin{cases} \max\left\{G(j+1, w),\ G(j+1, w-p_j) + p_j\right\}, & 1 \leq j \leq n \\ 0, & j = n+1 \end{cases} \tag{3.3}$$

The calculation of the values $G(j, w)$ is actually done in a backward fashion (concerning parameter j) because equation (3.3) requires already calculated values for $j+1$. So initially we set $G(n+1, w) := 0$ for a dummy customer $n+1$ and for all $w \in \{0, \ldots, W\}$, since we cannot increase the satisfied demand without having a customer left to append. For values of $G(j, w)$ with $1 \leq j \leq n$, Algorithm 3.1 needs to decide whether taking customer j into the knapsack (with weight restriction w) is advantageous or not. In the latter case we set $G(j, w) \leftarrow G(j+1, w)$ in step 8 because this decision implies that the best customer selection for weight restriction w has already been found in the iteration before. Otherwise, we can increase the "value of the knapsack" if we take an optimal set of orders as determined for $G(j+1, w-p_j)$ and add the demand p_j for customer j, which is done in step 10 of the algorithm.

After execution of procedure COMPUTEMAXGAINVALUES(S) the values of the variables $G(j, w)$, $j = 1, \ldots, n+1$ and $w = 0, \ldots, W$, are available to be retrieved by the branch and bound-algorithm. If the latter has arrived at a node v_σ with $\sigma = (\sigma_1, \ldots, \sigma_k)$ in the search tree, the above mentioned value G^*_σ can be determined by querying the variable $G\left(\sigma_k + 1, \delta_{max}\right)$. Hence, the described pre-computation allows to check the bounding

inequality (2.6) in constant time in each node of the search tree (if δ_{max} can be computed in constant time), whereas the running time of ComputeMaxGainValues(S) is $\mathcal{O}(nW)$, which is pseudo-polynomial in the size of the input.

This pre-computation can be regarded as a form of caching: Instead of solving a rather small knapsack problem in each node of the search tree, we solve a larger knapsack problem only once and reuse the values computed in the course of the dynamic programming algorithm. An advantage of this method on modern computers is that this calculation is supported by hardware caching facilities due to the principle of locality (Denning, 2005).

3.3 Delaying the production start

In this section we extend the basic model presented before by allowing **delayed production starts**. In general, this modification increases the number of feasible solutions, since a delayed production start will also make the orders expire later. Moreover, delaying the production start can produce better solutions for many problem instances because the product lifespan can be used for transportation rather than for waiting. Another practical motivation is that the introduction of production start delays makes it possible to consider latest delivery times for a given customer sequence, providing a time window for a feasible production start.

In the basic model of Section 2.1, the transportation stage begins at time $\tau_0 = \sum_{j \in \sigma} p_j$ for a given customer selection σ, which implies that the production of the first order in σ always starts at time 0. Note that we assume that the input values are scaled such that in each time unit exactly one unit of demand can be produced, that is, we may assume that the production rate R equals 1. Time 0 can be regarded as the earliest possible time for the production start. Especially when dealing with non-maximal selections, e.g. while incrementally constructing a schedule, there may be a waiting time w_{σ_1} for the delivery vehicle at the first customer σ_1 in σ. In this case, it is clearly reasonable to delay the beginning of production for at least w_{σ_1} time units, since this also delays the expiry dates of all orders, providing more freedom for planning while preserving time window feasibility.

In the following we denote by $ESS_\sigma(t)$ the **earliest start schedule** for a customer selection $\sigma \subseteq S$ and production start time $t \geq 0$. This schedule is obtained by beginning the production at time t, producing all orders in σ consecutively without idle times and then delivering these orders in the same sequence to the customers as early as possible, subject to travel times and time windows as described in Section 2.1.

i	σ_i	$[a_{\sigma_i}, b_{\sigma_i}]$	c_{σ_i}	τ_{σ_i}	w_{σ_i}	Δ_{σ_i}	l_{σ_i}
1	1	$[19, 24]$	3	19	3	0	0
2	3	$[31, 36]$	9	31	8	0	0
3	4	$[32, 35]$	12	35	0	0	1

Table 3.2: Values for selection $\sigma = (1, 3, 4)$

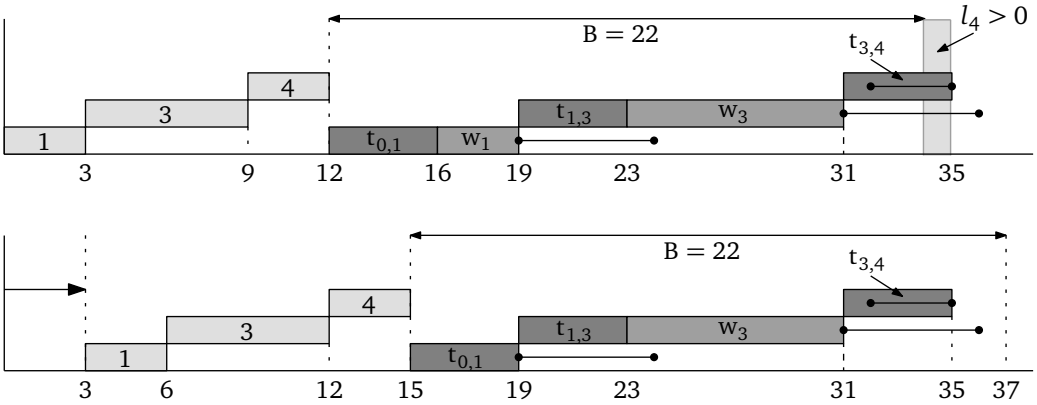


Figure 3.2: Delaying the production to obtain a feasible schedule for orders 1, 3 and 4

This implies that for $ESS_\sigma(t)$, the values $c_i, \tau_i, w_i, \Delta_i$ and l_i (cf. Table 3.2) are easily calculated in linear time for all $i \in \sigma$.

To illustrate the advantage of the above mentioned "right shifts" we consider Example 1 again, together with the customer selection $\sigma = (1, 3, 4)$. When computing the earliest start schedule $ESS_\sigma(0)$ for this selection, we obtain the values given in Table 3.2 and the corresponding schedule depicted in the upper part of Figure 3.2, which obviously obeys the time window constraints since $\Delta_j = 0$ for all $j \in \sigma$. However, the goods for customer 4 will have expired at delivery because $\tau_4 - c_4 = 35 - 12 = 23 > 22 = B$, which renders the schedule infeasible. In this case, the infeasibility can easily be repaired since the generated plan contains a waiting time $w_{\sigma_1} = w_1 = 3$ at the first customer. Thus, a delay of the production start of 3 time units makes the schedule become feasible again, which is illustrated in the lower part of Figure 3.2. Note that a shift of only 1 time unit is sufficient in this example.

Let $L_\sigma := \max_{j \in \sigma}\{l_j\}$ denote the **maximum lifespan tardiness** of all orders in σ. Delaying the production start by $w_{\sigma_1} > 0$ time units can only make the schedule become feasible if $w_{\sigma_1} \geq L_\sigma$. In the example we have $w_{\sigma_1} = 3 \geq 1 = L_\sigma$, so the given instance admits a feasible schedule.

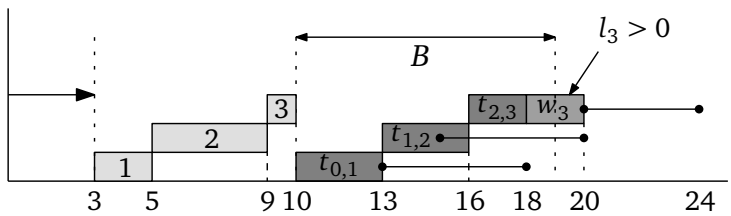


Figure 3.3: Consuming waiting times to avoid lifespan tardiness

Now consider the situation shown in Figure 3.3. We assume that in this schedule a former waiting time w_1 of 3 time units for the first customer has already been removed by delaying the production start until time 3, such that order 1 (and 2) can be delivered without any waiting time. But there is still a waiting time $w_3 > 0$ for the third customer after the shift. With regard to the assumed product lifespan, the goods contained in the third order will expire while the vehicle is waiting for the beginning of the delivery time window at time 20. Thus, the considered selection is still infeasible. Nevertheless, we can deliver all goods in time for this selection by simply delaying the production start for one more time unit. This is possible due to the slack in the time windows for customers 1 and 2. As a result of the shift, orders 1, 2 and 3 are then delivered at times 14, 17, and 20, respectively. These delivery dates now all obey the lifespan constraints and we obtain a feasible schedule.

We will now formulate a criterion to decide whether a lifespan-infeasible schedule may be transformed into a feasible one by simply delaying the production start. The following recursive definition of the *delaying potential* of an order σ_i implies a method to compute the maximum delay for the delivery of order σ_i such that its respective deadline and the deadlines of all subsequent orders will still be met.

Definition 2 *Let $\sigma = (\sigma_1, \ldots, \sigma_k)$ be a selection of orders such that the corresponding schedule $ESS_\sigma(0)$ obeys all time window constraints, i.e. $\Delta_{\sigma_i} = 0$ for all $\sigma_i \in \sigma$. The* ***delaying potential*** *δ_{σ_i} of an order $\sigma_i \in \sigma$ is recursively defined by*

$$\delta_{\sigma_i} := w_{\sigma_i} + \min\left\{b_{\sigma_i} - \tau_{\sigma_i}, \delta_{\sigma_{i+1}}\right\} \quad \text{for } i = k, \ldots, 1, \tag{3.4}$$

where initially $\delta_{\sigma_{k+1}} := \infty$ for a dummy order σ_{k+1}. The ***delaying potential of the selection σ*** *is defined as the delaying potential of the first order in σ: $\delta_\sigma := \delta_{\sigma_1}$.*

In the above definition we assume that we are given a time window feasible schedule, i.e. for all $i \in \sigma$ we have $\tau_i \leq b_i$. This is important for the definition because otherwise we would obtain negative delays, rendering the sequence infeasible. We can compute

the values $\delta_{\sigma_k}, \delta_{\sigma_{k-1}}, \ldots, \delta_{\sigma_1}$ for a given schedule in time $\mathcal{O}(k)$. Finally, the value δ_σ represents the maximum possible delay for the delivery of the first order (with respect to all subsequent orders), and thus for the start of the entire production. Note that the definition of the delaying potential is very similar to the maximum transport delay in Definition 1. The main difference is that for the delaying potential we do not need to consider lifespan constraints because delaying the production start cannot incur further lifespan violations, since expiry dates are also delayed.

As a positive tardiness $l_{\sigma_i} > 0$ for an order $\sigma_i \in \sigma$ can only by removed by decreasing the duration between the end of its production c_{σ_i} and the respective delivery time τ_{σ_i}, there must be sufficient waiting time before the delivery of σ_i that can be removed by delaying the production start. In other words, if there are no waiting times in the current schedule but a positive tardiness occurs, there is no way to eliminate it, although $\delta_\sigma > 0$. In this situation no feasible schedule can be obtained. Let $W_{\sigma_i} := \sum_{j=1}^{i} w_{\sigma_j}$ denote the *cumulative waiting time* before the delivery of order σ_i. Then a necessary condition for removing a positive tardiness $l_{\sigma_i} > 0$ at a customer σ_i is given by $W_{\sigma_i} \geq l_{\sigma_i}$. We use this condition in the following theorem which offers a necessary and sufficient condition for removing all violations of lifespan restrictions in a time window feasible schedule by delaying the production start.

Theorem 3 *Let $\sigma = (\sigma_1, \ldots, \sigma_k) \subseteq S$ be a customer selection, whose associated earliest start schedule $ESS_\sigma(0)$ obeys all time window constraints, i.e. $\Delta_{\sigma_i} = 0$ for all $\sigma_i \in \sigma$. The schedule obtained by delaying the production start by the delaying potential δ_σ is feasible with respect to both time window and lifespan constraints if and only if*

$$\delta_\sigma \geq L_\sigma \quad \textit{and} \quad W_{\sigma_i} \geq l_{\sigma_i} \; \forall \, \sigma_i \in \sigma. \tag{3.5}$$

Proof At first we will show that the schedule which is obtained by delaying the production start by δ_σ time units is feasible when (3.5) is satisfied. Since, by definition, δ_σ is the largest possible delay for the first delivery in σ while maintaining time window feasibility for all orders, the schedule is time window feasible. Assume to the contrary that the obtained schedule is not lifespan feasible, i.e. there is at least one order σ_i with a positive tardiness $l'_{\sigma_i} > 0$ (a prime indicates the new values after the shift). So this shift cannot sufficiently decrease the duration of the (new) interval $[c'_{\sigma_i}, \tau'_{\sigma_i}]$ to allow customer σ_i to be served before the expiry of the goods. This may be due to two reasons: firstly, the occurring tardiness may be too large to be compensated, i.e. $\delta_\sigma < l_{\sigma_i} \leq L_\sigma$. However, this contradicts the first part of condition (3.5). Secondly, under the assumption that $\delta_\sigma \geq L_\sigma$, there must be insufficient waiting time to be consumed before τ_{σ_i}. This implies $W_{\sigma_i} < l_{\sigma_i}$, contradicting the second part of condition (3.5).

For the reverse implication, assume that the original schedule has been delayed by δ_σ time units, producing a feasible schedule. We have to show that condition (3.5) holds. First, if we had $\delta_\sigma < L_\sigma$, then there would have been no way to remove the occurring tardiness of L_σ time units by a right shift and thus no feasible schedule could have been obtained. This contradicts the precondition. Second, if $\delta_\sigma \geq L_\sigma$ and there was an order $\sigma_i \in \sigma$ in the original schedule with $W_{\sigma_i} < l_{\sigma_i}$, then the shift could not have removed the tardiness at σ_i since there would have been insufficient waiting time before τ_{σ_i} in the schedule to reduce the duration between τ_{σ_i} and c_{σ_i}. Thus, no feasible schedule could have been obtained, again contradicting the precondition. □

Definition 4 *A customer selection $\sigma \subseteq S$ is called **shift-feasible** if a production start time $t \geq 0$ exists such that the associated earliest start schedule $ESS_\sigma(t)$ is feasible.*

Now, if we are given a shift-feasible selection $\sigma \subseteq S$, we can precisely determine a time window $\mathcal{F}_\sigma$, within which the production phase must begin in order to obtain a feasible schedule for σ.

Theorem 5 *Let $\sigma \subseteq S$ be a shift-feasible customer selection. Then the earliest start schedule $ESS_\sigma(t)$ is feasible if and only if $t \in \mathcal{F}_\sigma := [L_\sigma, \delta_\sigma]$.*

Proof By assumption, $ESS_\sigma(\delta_\sigma)$ is feasible, because σ is shift-feasible and δ_σ is the largest possible shift of the production start. Now, consider the case $t = L_\sigma$. Since $L_\sigma \leq \delta_\sigma$ by Thm. 3, $ESS_\sigma(L_\sigma)$ does not violate the time window restrictions. It remains to show that $ESS_\sigma(L_\sigma)$ also obeys the lifespan restrictions. Again due to Thm. 3, we have $W_{\sigma_i} \geq l_{\sigma_i}$ for each $\sigma_i \in \sigma$. Let $s \in \sigma$ be the frontmost order in the sequence, for which $l_s = L_\sigma$. Then, a shift of the production start by $t = L_\sigma$ will "compress" the waiting times before customer s, such that all lifespan violations occurring before s will be removed by the shift. Due to the consumption of waiting times, the delivery time for customer s will not change. Consequently, all subsequent delivery times will not change, either. Since the tardiness of each order after s is at most $l_s = L_\sigma$ by choice of s, each such tardiness will also be removed and schedule $ESS_\sigma(L_\sigma)$ is feasible w.r.t. lifespan and time window constraints. Furthermore, for each t with $L_\sigma < t < \delta_\sigma$ the schedule $ESS_\sigma(t)$ is also feasible because this shift cannot cause further lifespan violations nor disobey the time window constraints. □

The results of this section are useful for implementing feasibility checks for a given selection σ by Theorem 3. The feasibility interval for the production start $\mathcal{F}_\sigma$ introduced in Theorem 5 can be computed in linear time $\mathcal{O}(|\sigma|) = \mathcal{O}(n)$. It is useful for computing earliest- and latest start schedules for a given customer selection. Thus, it

can easily be decided whether a feasible schedule exists when delaying the production start is allowed.

The original mixed integer programming model (2.1) (see p. 18) can also easily be adapted to reflect the model extension discussed in this section. We need to introduce a further variable $\delta \geq 0$, which is added to the left hand sides of each of the time window constraints (2.1b) and (2.1c), respectively. Additionally, a weighting parameter $\alpha \in \mathbb{R}$ can be used to specify a preference for the production delay in the objective function, which must be altered in the following way:

$$\text{Maximize} \quad P_{\text{shift}} = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} p_j Y_{i,j} + \alpha\delta \tag{2.1a$'$}$$

Choosing $\alpha > 0$ yields the largest feasible value for δ in the computed solution, whereas choosing $\alpha < 0$ yields the smallest value for δ. However, $|\alpha|$ must not be too large because then a MIP-solver could possibly exclude a customer in an optimal solution in exchange for achieving a larger right-shift (larger δ). Using a value for α that obeys the condition $|\alpha| \leq \frac{\min_i\{p_i\}}{\max_i\{b_i - t_{0,i}\}}$ produces a correct result, since for such a value it is never beneficial (in terms of a larger objective function value) to delay the production phase for one time unit instead of supplying another customer.

3.4 Tabu search for fixed customer sequences

Our analysis of the branch and bound algorithm described in Section 2.2 with the extension of production delays shows that this exact method is capable of solving instances of up to 50 customers to optimality. This threshold is even lower in case of very short lifespans. For larger instances, however, heuristic methods must be employed to obtain good solutions in reasonable runtimes. Armstrong et al. (2008) present a simple deletion-based heuristic for the basic problem (without shifts) that starts with a complete selection of all possible customers and iteratively removes suitable pairs of customers in a greedy fashion until a feasible solution is found. This heuristic can easily be extended for the model with shifts because we only need to integrate appropriate feasibility checks. Nevertheless, this solution method has been shown in Viergutz and Knust (2010) to yield inferior results especially for short lifespans and/or narrow customer time windows. This is mainly due to its greedy approach that only allows *deleting* customers until a feasible solution is obtained, without considering possible re-addition steps.

A well known paradigm to improve given solutions heuristically is called **Local Search (LS)**, which is an iterative procedure that explores the set of feasible solutions $\mathcal{S}$ by moving from a given solution $s \in \mathcal{S}$ to another solution $s' \in \mathcal{S}$ such that s' is a member of a predefined neighborhood $\mathcal{N}$ of s, denoted by $s' \in \mathcal{N}(s)$. The neighboring solutions typically share many similarities, for example, in the scheduling problem considered here, s and s' may be two customer selections where s' contains the same subset of customers as s, with only one more customer added, or vice versa. For this reason, a neighborhood in LS is often specified by a set of operators that describe possible modifications of a solution $s \in \mathcal{S}$. Obvious operators for our scheduling problem include adding a previously unselected customer or replacing a currently selected customer by an unselected one. LS repeatedly applies such operators to the current solution and chooses a new current solution for the next iteration based on the observed objective function values until a stopping criterion is satisfied. Different types of LS can be obtained by applying different criteria for choosing the next solution. A very simple LS method is called iterative improvement (or hill climbing), where in each iteration a neighbor solution with a better objective function value than the current is chosen, as long as one exists. This way, LS finds (only) a locally optimal solution (none of its neighbors has a better objective function value), which may, however, be substantially worse than a globally optimal solution. In order to escape from local optima, heuristics need to be able to make moves to solutions having worse objective function values than the current solution. Unfortunately, this entails the problem of re-visiting already found solutions, which may cause cycling, i.e. failure to terminate.

In this section we propose a variant of **Tabu Search (TS)**, which is a metaheuristic that was first proposed by Glover (1986), and further developed by contributions of Hertz and de Werra (1990). It is based on local search and employs adaptive memory to prevent the search process from cycling. This adaptive memory is organized as a **tabu list**, where already visited solutions (or merely certain attributes thereof) are stored. These solutions are called **tabu** and are not considered as candidate solutions the search can move to. In general, due to memory restrictions TS cannot store all visited solutions during the search process. Hence, the length of the tabu list must be limited. This implies that TS can only prevent cycles of length at most the size of the tabu list. However, with an increasing tabu list size the probability of cycling becomes sufficiently small. A frequent approach in TS methods consists in storing only typical solution properties (also called attributes) in the tabu list, instead of storing full descriptions of the solutions. In this case, all solutions having one of the stored attributes are considered tabu. The goal is to find suitable properties that are able to prevent the search from re-visiting solutions during the period in which the corresponding property is stored in the tabu list.

Algorithm 3.2 Tabu search for customer selections

1: **procedure** TABUSEARCHFORSELECTIONS(S)
2: Compute a feasible starting selection $\sigma \subseteq S$
3: Initialize tabu list $TL \leftarrow \emptyset$
4: Initialize $\sigma^* \leftarrow \sigma$ and $p^* \leftarrow p_\sigma$
5: **repeat**
6: **if** $\sigma = S$ **then return** σ
7: **if** all unselected orders are tabu **then** $TL \leftarrow \emptyset$
8: Choose customer $i \in S \setminus \sigma$ that is non-tabu or meets an aspiration crit.
9: Append customer i to sequence σ
10: **if** σ is not (shift-) feasible **then**
11: $E \leftarrow \{j \mid \sigma \setminus \{j\}$ is (shift-) feasible, $j \neq i\}$
12: **if** $E \neq \emptyset$ **then**
13: Choose a customer $k \in E$ having smallest demand p_k
14: Remove k from σ and append k to TL
15: **else**
16: Remove i from σ and append i to TL
17: **if** $p_\sigma > p^*$ **then**
18: $p^* \leftarrow p_\sigma$ and $\sigma^* \leftarrow \sigma$
19: **until** a stopping condition is satisfied
20: **return** σ^*

A suitable and concise representation for a customer selection σ is given by a bit vector $b(\sigma) = (b_1, \ldots, b_n) \in \{0,1\}^n$, where $b_i = 1$ if and only if customer i is selected. The set-related operations in the procedure TABUSEARCHFORSELECTIONS (see Algorithm 3.2) can easily be implemented using this bit vector representation. If $\sigma \subseteq S$ is a customer selection that admits a feasible schedule, we denote by p_σ the corresponding total satisfied demand, i.e. $p_\sigma := \sum_{j \in \sigma} p_j$.

The procedure TABUSEARCHFORSELECTIONS is actually generic because it does not specify directly how to obtain a feasible starting selection σ (step 2) or how to choose a customer for insertion (step 8). Starting selections may simply be generated by adding customers successively to the empty selection in a greedy fashion, i.e. ordered by non-increasing demands, until no more customers can be added without making the selection infeasible. Of course, a random starting solution (adding random customers as long as the selection remains feasible) is also possible, as well as using the result of the above mentioned deletion heuristic by Armstrong et al. The customer selection in step 8 can be implemented similarly, i.e. we can try to include a random unselected customer or an unselected customer having the largest demand. In our computational experiments, the combination of greedy starting solutions and the random strategy for customer selections showed the best results.

Algorithm 3.2 contains a feasibility check for the current sequence σ in step 10, thus it is easy to adapt the tabu search to scenarios with and without shifts of the production start by using the appropriate checking method. The body of the repeat-loop ends at step 19, where the loop terminates if a predefined stopping condition is satisfied. This condition might be a combination of several criteria, e.g. a maximum number of iterations, a runtime limit or the lack of improvement of the best known solution in a specified number of iterations.

In TabuSearchForSelections a tabu list TL is maintained that is limited to a maximum length of $\lambda \in \mathbb{N}$ entries. This list is organized as a FIFO-queue of customers which either have recently been removed from the selection (line 14) or could not be feasibly included into the sequence in one of the last λ search steps (line 16). Customers contained in TL are not considered for insertion into the selection in the current iteration.

The **tabu tenure** of each customer i in the tabu list is determined by the value of λ at the time i is inserted. As stated by Glover and Laguna (1993, p. 80), "improved outcomes are often obtained by tabu tenures that vary dynamically". Furthermore, the authors argue that "practical experience indicates that dynamic rules [for tabu tenures] are typically more robust than static rules" (p. 97). For this reason we have chosen to employ an adaptive, dynamic tabu tenure rule for the elements of TL depending on the recently observed values of the objective function. If the search process is in an **improving phase**, i.e. the value of the objective function has increased during the last iteration, then we set $\lambda := \max\{\lambda - 1, \lambda_{min}\}$. Otherwise, in a **deteriorating phase** of the search, we set $\lambda := \min\{\lambda + 1, \lambda_{max}\}$. The parameters λ_{min} and λ_{max} are kept constant over the entire tabu search and ensure that $\lambda_{min} \leq \lambda \leq \lambda_{max}$ in each iteration. The underlying idea of this method is that fewer customers should be considered tabu in an improving search phase since this assists **intensification**. When deteriorating moves occur, tabu tenures are increased up to a value of λ_{max}, which constrains the search path in order to quickly escape from unfavorable regions of the search space.

As a means of **diversification**, our tabu search performs random restarts whenever there has been no improvement of the best known solution in a pre-specified number n_{div} of search steps. Tests showed that a value of $n_{\text{div}} = 200$ was suitable for most instances of the considered scheduling problem. Another commonly used technique in tabu search algorithms is called *aspiration criteria*, which are designed to decide under which circumstances the tabu status of a solution should be overridden. In TabuSearchForSelections we include an anspiration criterion at line 8, where we allow a customer $i \in TL$ to be included whenever this leads to schedule with a better objective value than that of the incumbent (best known) solution.

3.5 Alternative objective functions

The basic model aims at maximizing the total satisfied demand that can feasibly supplied to the given set of customers. Of course, this is not the only possible objective function for the considered integrated production and distribution problems. In the following we will discuss the objective of minimizing the costs of transportation, which is especially interesting when variable customer sequences are allowed (see chap. 4). Furthermore, if latest delivery times (from customer time windows) are treated as soft deadlines which may be violated at a penalty, we may be interested in minimizing the (weighted) delivery tardiness.

3.5.1 Transportation costs

Another widely used component in similar models consists in the minimization of transportation costs, such as fixed costs for the deployment of the transport vehicle(s) and the total travel time (routing costs). These costs can only constitute the entire objective function if a further constraint ensures that a given minimum amount of demand is satisfied by the deliveries. However, this leads to a different model, which is similar to the so-called prize-collecting traveling salesman problem (PCTSP), where the goal is to find a circuit that minimizes travel costs and whose associated satisfied demand is not smaller than a preset value (Feillet et al., 2005; Awerbuch et al., 1999). Admittedly, the PCTSP is more complex due to the inherent sequencing subproblem, which is not part of our model so far. Nevertheless, we will discuss the model extension of allowing variable production and delivery sequences in Chapter 4. It is obvious that routing costs play a more important role in a setting like that.

The most straightforward way of incorporating the minimization of travel costs in our model consists in substracting those costs, multiplied by a controlling parameter $\beta \geq 0$, from the existing objective function. In this case, equation (2.1a) must be adapted in the following way:

$$\text{Maximize} \quad P_{\text{transport}} = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} p_j Y_{i,j} - \beta \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} t_{i,j} Y_{i,j} \tag{2.1a''}$$

The parameter β must be chosen carefully depending on the considered problem instance, because, on the one hand, if the value of β is too large, solvers will tend to serve less customers in order to obtain a shorter distribution tour. On the other hand, too small values of β will show hardly any effect.

3.5.2 Weighted delivery tardiness

A basic requirement for a feasible production and distribution schedule according to our current model is that each delivery must take place during the predefined customer time windows. In the literature, many articles distinguish between hard and soft time windows (Taillard et al., 1997; Bräysy and Gendreau, 2002), where the term "soft time windows" describes a setting in which delivery deadlines are not mandatory and late services (after the end of the resp. time window) are allowed. Services that occur outside the specified time windows are then penalized with a preset non-negative weight in the objective function. Hence, using soft time windows essentially represents a relaxation of the scenario with hard time window constraints. The latter are given by inequalities (2.1b) and (2.1c) on page 19.

Using weights also allows us to state preferences concerning the customers, for example that certain customers have a higher priority for receiving their goods on time. The determination of appropriate weights represents a problem of its own, since using very large weights will force solvers to generate solutions that are (at least almost) also feasible for the hard-time-window scenario, whereas the influence of the remaining part of the objective function (maximizing total satisfied demand) is nearly negligible.

In order to formally describe the required changes to the existing objective function of MIP (2.1), we introduce the notation $\Delta_k := \max\{0, \tau_k - b_k\}$ for the tardiness of an order $k \in \{1, \ldots, n\}$, where

$$\tau_k := \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} p_j Y_{i,j} + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k} t_{i,j} Y_{i,j} + \sum_{i=1}^{k} w_i$$

represents the delivery time at customer k in terms of the variables contained in the MIP formulation. We use the derived variables Δ_k as an abbreviation for the respective differences of right- and left-hand-sides of inequality (2.1c). If any of these variables has a strictly positive value for an order k the corresponding delivery happens outside of the specified time window $[a_k, b_k]$.

A problem with the previous definition(s) is that the values Δ_k, and τ_k are not well-defined for an unselected customer k. However, simply changing the basic model by adding the requirement that each customer *must* be supplied – as it is common in models for the traveling salesman problem with time windows (TSPTW) – does not lead to an interesting problem here due to the fixation of the delivery sequence. For variable delivery sequences (as we will discuss in Chapter 4), this requirement, together with the weighted tardiness objective, yields more attractive optimization

models. Returning to the basic model presented so far, we can incorporate tardiness penalties in the objective function by introducing further integer variables $Z_k^\Delta \geq 0$ for each customer $k = 1, \ldots, n$, such that $Z_k^\Delta = 0$ if k is unselected, and $Z_k^\Delta = \Delta_k$ if k is selected. Note that a customer k is selected in the basic model if and only if $\sum_{i=0}^{k-1} Y_{i,k} = 1$. The described relationship between the variables can be established by adding the constraints

$$Z_k^\Delta = \Delta_k \sum_{i=0}^{k-1} Y_{i,k} \quad (k = 1, \ldots, n) \tag{3.6}$$

But, unfortunately, the term on the right-hand side is non-linear. The following demonstrates how to rewrite these constraints via three linear restrictions. For each $k = 1, \ldots, n$ we add the following constraints, using sufficiently large numbers M_1, M_2, and M_3 that represent upper bounds of the respective left-hand sides of the inequalities.

$$Z_k^\Delta \leq M_1 \sum_{i=0}^{k-1} Y_{i,k} \tag{3.7}$$

$$Z_k^\Delta - \Delta_k \leq M_2 (1 - \sum_{i=0}^{k-1} Y_{i,k}) \tag{3.8}$$

$$\Delta_k - Z_k^\Delta \leq M_3 (1 - \sum_{i=0}^{k-1} Y_{i,k}) \tag{3.9}$$

Now, if customer k is selected, i.e. if $\sum_{i=0}^{k-1} Y_{i,k} = 1$, restrictions (3.8) and (3.9) enforce $Z_k^\Delta = \Delta_k$, while (3.7) is trivially satisfied. Otherwise, if k is unselected, (3.7) enforces $Z_k^\Delta = 0$, while (3.8) and (3.9) are trivially satisfied. Using these additions, we can state the adapted objective function for weighted tardiness by introducing non-negative weights $\vartheta_k \geq 0$ for each customer $k = 1, \ldots, n$. Since we are considering a maximization problem, we need to subtract the penalty costs for tardy deliveries from the total satisfied demand:

$$\text{Maximize} \quad P_{\text{tardy}} = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} p_j Y_{i,j} - \sum_{k=1}^{n} \vartheta_k Z_k^\Delta \tag{2.1a'''}$$

As already explained before, the choice of the weights ϑ_k has a strong influence on the obtained optimum. On the one hand, using large weights will simulate the setting of hard time windows, because then customers will rather be left unsupplied instead of being supplied with a large tardiness. On the other hand, solutions for small weights will tend to serve more customers, at the expense of paying penalty costs.

Chapter 4

Variable production and distribution sequences

In this chapter, we introduce an extension of the basic integrated production and distribution model, where variable customer sequences are allowed. Not only do the discussed extensions provide a more realistic modeling environment, but they also permit better solutions in many cases, i.e. the delivery vehicle is able to serve more customers or, to be more precise, to satisfy more demands in the same planning period compared to the fixed-sequences scenario.

4.1 Models and motivation

For many practical applications, the assumption of a predefined, fixed delivery sequence in the basic model of Section 2.1 appears overly restrictive. Since customer time windows and travel times are usually not related, extending the model by introducing variable production and delivery sequences can be advantageous. Thus, in this chapter a customer selection $\sigma \subseteq S$ is always considered in combination with a sequence for all elements in σ. In the following, we will treat the terms "customer sequence" and "customer selection" as synonyms, as long as this is no cause for confusion.

Let σ^P denote the production sequence and let σ^D denote the distribution sequence. Now, there are basically two scenarios for this model generalization:

1. A single sequence σ (to be calculated by a solution algorithm) determines the production and distribution sequence, i.e. $\sigma^P = \sigma^D = \sigma$.

2. Production and distribution sequences are allowed to be different, i.e. the delivery sequence σ^D is only required to be a permutation of the production sequence σ^P.

For the first scenario, a potential heuristic solution approach consists in trying different sequences σ (e.g. obtained via local search) in an "outer loop" and solving the associated problem for fixed sequences by the algorithms presented for a fixed customer sequence.

In general, the model becomes more difficult to solve when the second scenario is considered. However, tying the delivery sequence to the production sequence lacks practical justification and can prevent better objective function values. The following Example 2 shows that in the case of a limited product lifespan $B < \infty$ there are problem instances for which a better schedule (in terms of total satisfied demand) can be obtained by choosing different production and distribution sequences.

(a) Travel times

t_{ij}	1	2
0	3	1
1		2

(b) Demands and time windows

Customer j	1	2
p_j	4	1
a_j	6	6
b_j	8	9

Table 4.1: Parameters for Example 2

Example 2 We consider a small problem instance with $n = 2$ customers and the parameters shown in Table 4.1. Note that the travel times obey the triangle inequality. Moreover, let the product lifespan be $B = 4$. There may possibly be 4 different production and distribution schedules if both customers must be supplied. These schedules are depicted in Figure 4.1 (a)–(d). In part (a), orders are both produced and delivered according to the sequence $\sigma^P = \sigma^D = (1, 2)$. The resulting delivery times are $\tau_1 = 8$ and $\tau_2 = 10$, which renders the plan infeasible because firstly, the time window is missed for customer 2 ($\Delta_2 = 1$) and secondly, the lifespan is exceeded at that customer, causing a positive tardiness $l_2 = 1$. Analogous arguments show that schedules (c) and (d) are also infeasible. The remaining schedule (b) corresponds to the production sequence $\sigma^P = (1, 2)$ and the delivery sequence $\sigma^D = (2, 1)$, which yields the only feasible schedule where both customers are served. The delivery times are $\tau_2 = 6$ and $\tau_1 = 8$, which are both within the given time windows and the lifespan of the product. $\Diamond$

The example demonstrates that decoupling the order in which the goods are produced from the order in which they are delivered can yield better schedules where possibly

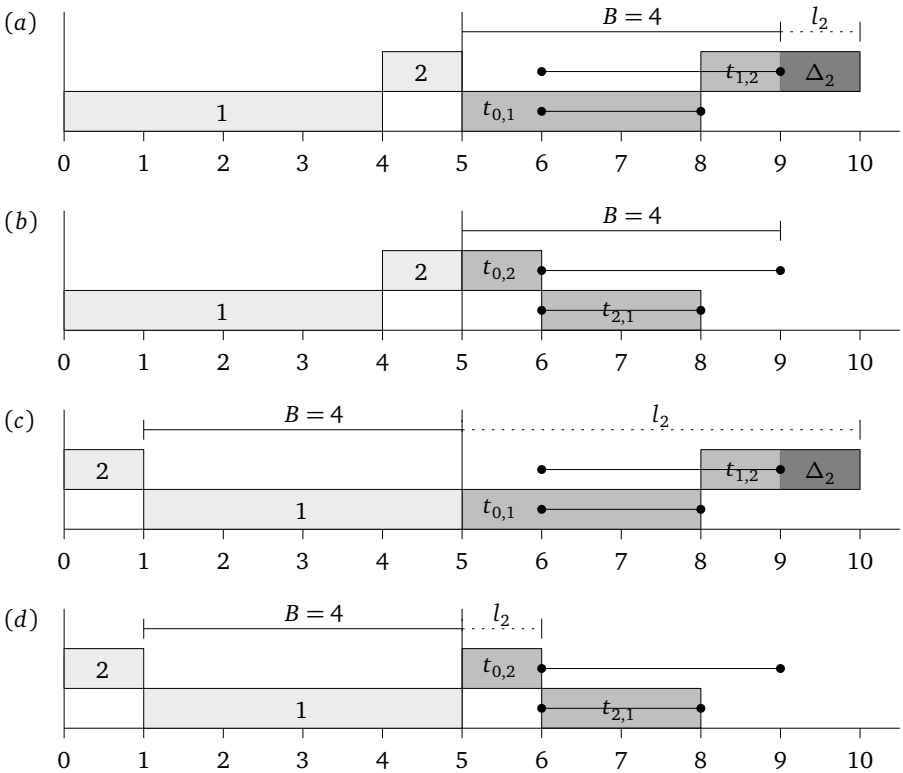


Figure 4.1: All possible (complete) schedules for the instance of Example 2: Only the schedule shown in (b) is feasible (and thus optimal), where production- and distribution sequences are different.

more customers can be served feasibly. However, in situations where the product lifespan B is unlimited (or very large compared to travel and production times) the following theorem shows that there is always an optimal schedule whose production and distribution sequences are equal.

Theorem 6 *If the lifespan is infinite, there is always an optimal production and distribution plan where the goods are produced in the same order as they are delivered.*

Proof Suppose that for a given problem instance there is an optimal production and distribution plan $\hat{\sigma} = (\hat{\sigma}^P, \hat{\sigma}^D)$, in which $\hat{\sigma}^P$ differs from $\hat{\sigma}^D$. Since the product lifespan is assumed to be infinite, we can simply reorder the production sequence to match the distribution sequence $\hat{\sigma}^D$. The resulting plan $\hat{\sigma}' = (\hat{\sigma}^D, \hat{\sigma}^D)$ is feasible because reordering the production sequence does not change the start time of the

distribution phase. Thus, $\hat{\sigma}'$ is also an optimal plan with equal production and distribution sequences. □

Theorem 6 confirms that it might only be beneficial to study the second scenario (where $\sigma^P \neq \sigma^D$ is possible) if lifespan conditions are actually restrictive, which is, however, assumed in most cases that are discussed in this thesis.

Complexity Adding a sequencing subproblem in the first scenario or even two in the second scenario makes the resulting production and distribution scheduling problem generally more difficult to solve than the basic problem of Chapter 2, where the production and distribution sequences were assumed to be equal and fixed in advance.

Theorem 7 *The integrated production and distribution scheduling problem with variable customer sequences is $\mathcal{NP}$-hard. It remains $\mathcal{NP}$-hard even if the product lifespan is unlimited.*

The theorem can be proved by a simple polynomial-time reduction from the feasibility version of the traveling salesman problem with time windows (TSPTW) – known to be $\mathcal{NP}$-complete (Garey and Johnson, 1979) – to the decision version of the considered variable sequences problem: *Is there a feasible production-distribution schedule with a total satisfied demand of at least P?*

Firstly, it is plain to see that the latter problem belongs to $\mathcal{NP}$, since, given two customer sequences σ^P and σ^D and a production start time, it is easy to check in polynomial time whether the sequences are based on the same subset of customers and whether the resulting earliest start schedule (ESS) is feasible with respect to time windows and lifespan restrictions. Secondly, to briefly sketch the required transformation, in the transformed instance we set the lifespan $B = \infty$, the production rate $R = 1$, the demand $p_i = 1$ for each customer i (node in the TSPTW graph other than the depot), and the time windows $[a'_i, b'_i] := [a_i + n, b_i + n]$, where $[a_i, b_i]$ are the time windows from the TSPTW instance and n represents the number of customers. Furthermore, we need to set the demand goal to $P := n$. Then, there is a feasible solution for the TSPTW instance if and only if the transformed instance for the variable sequences problem admits a schedule that yields a satisfied demand of at least P.

Since, for the transformed instance in the above reduction, we can set the lifespan B to infinity independent of the original TSPTW-instance, it is clear that the integrated production and distribution problem is already $\mathcal{NP}$-hard in the relaxed version, where lifespans are not considered.

4.2 Equal production and distribution sequences

In this section we discuss the **single vehicle, equal sequences problem (SVESP)**. It consists of simultaneously finding a selection and a sequence of the customers $\sigma = \sigma^P = \sigma^D$ so that a maximum amount of the product can be delivered, subject to time windows and product-lifespan constraints. This setting was introduced in the previous section as scenario 1 and it requires the goods to be delivered in the same order as they are produced at the facility. As previously shown in Theorem 6, there is always an optimal schedule with this property if the product lifespan is not very restrictive. However, even for many instances with rather short lifespans it is intuitively clear that using the same sequences for production and distribution represents a reasonable approach because this also makes the goods expire in the same sequence. So, at least from a heuristic viewpoint, high quality solutions to this problem also provide good starting solutions for the extended problem, where $\sigma^P \neq \sigma^D$ is allowed. The latter problem will be discussed in Section 4.3.

We present a mixed integer programming model for the SVESP and illustrate necessary extensions for the case when $\sigma^P \neq \sigma^D$ is permitted. Furthermore, we present heuristics to calculate good feasible solutions.

4.2.1 A mixed integer programming formulation

We introduce the following notation that is used in the mixed integer programming formulation for the SVESP:

- $C := \{1, \ldots, n\}$, the set of **customers**
- $N := C \cup \{0, n+1\}$, the **nodes of the transport graph**, including all customers in C and two additional nodes for the start- and end-depot, respectively. Start- and end-depot represent the same location, but they are modeled by two different nodes, i.e. 0 and $n+1$.
- $A := \{(i,j) \mid 0 \leq i \leq n,\ 1 \leq j \leq n+1,\ i \neq j\} \subset N \times N$, the set of **arcs of the transport graph**. We introduce arcs between each pair of nodes, but neither ingoing arcs for node 0 (source) nor outgoing arcs for node $n+1$ (target).

In order to restrict the number of variables that will correspond to the use of arcs in our model, the following preprocessing steps are often reasonable.

Preprocessing

For many problem instances, the size of the arc set A may be reduced by taking customer time windows into account. Often it is possible to remove an arc (i, j), because it is impossible to travel from customer i to j without violating time windows. Let us define the **earliest arrival time from** i **to** j as $EAT(i, j) := \max\left\{a_i + t_{i,j},\ a_j\right\}$. Then, the set of **successors** of a customer $i \in C$ is given as follows:

$$\mathcal{S}(i) := \left\{j \in N \mid EAT(j, i) > b_i\right\} \cup \{n + 1\}. \tag{4.1}$$

This definition ensures that $\mathcal{S}(i)$ contains all nodes j such that i cannot be reached from j before the end of i's time window. Then, in any feasible schedule where i and j are both supplied, j cannot precede i, i.e. i must precede j in this case. Additionally, finishing the tour by traveling to the terminal node $n + 1$ is always possible. Now, if $j \in \mathcal{S}(i)$ for any two customers $i, j \in C$, then the arc (j, i) can be removed from A, since, if i and j are both included in a selection, i must precede j and the arc (j, i) cannot be used in any feasible solution. For a given pair of customers $i, j \in C$ it is possible that i cannot be feasibly reached from j and vice versa. More formally, we can define the following set of **mutually exclusive (mutex) customers**:

$$\mathcal{M} := \left\{\{i, j\} \in \mathcal{P}_2(C) \mid i \in \mathcal{S}(j) \wedge j \in \mathcal{S}(i)\right\} \tag{4.2}$$

In this definition, $\mathcal{P}_2(C)$ denotes the set of all two-element subsets of C. Thus, if $\{i, j\} \in \mathcal{M}$, then it is impossible to construct a feasible schedule that contains both i and j. This information can be used to derive upper bounds for the scheduling problem: If total satisfied demand is considered as the objective function (like in equation (4.5a)), a trivial upper bound is given by the sum of demands of all customers $UB := \sum_{k \in C} p_k$. This bound can be strengthened by the following formula:

$$\begin{aligned} UB' &:= \min_{\{i,j\} \in \mathcal{M}} \left\{ \sum_{k \in C} p_k - \min\left\{p_i, p_j\right\} \right\} \\ &= \sum_{k \in C} p_k - \max_{\{i,j\} \in \mathcal{M}} \left\{ \min\left\{p_i, p_j\right\} \right\} \end{aligned} \tag{4.3}$$

UB' is a valid upper bound because for each pair $\{i, j\} \in \mathcal{M}$ at most $\max\left\{p_i, p_j\right\}$ demand units can be delivered, thus the initial upper bound UB may be reduced by $(p_i + p_j) - \max\left\{p_i, p_j\right\} = \min\left\{p_i, p_j\right\}$. If we use the best (i.e. smallest) bound computed in this way, we obtain UB'.

In fact, we can strengthen the latter bound UB' even further by using the following observation. We may think of the pairs $\{i, j\} \in \mathcal{M}$ as edges of an undirected graph

$G_{\mathcal{M}} = (\mathcal{V}, \mathcal{M})$, where each node (customer) i is weighted by the associated demand p_i. Then, the problem of finding a subset of the customers in $\mathcal{V}$, who can be feasibly included in the same production and distribution schedule and whose total demand is maximum, is equivalent to solving a *maximum weighted independent set problem* (MWIS) on $G_{\mathcal{M}}$. Since this problem is known to be $\mathcal{NP}$-hard (see Sakai et al., 2003), heuristics should be applied if $|\mathcal{M}|$ is large. However, most instances we examined in our computational evaluation contained a rather small number of mutexes. Thus, exact solutions to the MWIS problem could be obtained. Let $\alpha(G_{\mathcal{M}})$ denote the maximum weight of any independent set in $G_{\mathcal{M}}$, then the following value represents an upper bound for the scheduling problem:

$$\begin{aligned} UB'' &:= \sum_{k \in C} p_k - \left(\sum_{k \in \mathcal{V}} p_k - \alpha(G_{\mathcal{M}}) \right) \\ &= \sum_{k \in C \setminus \mathcal{V}} p_k + \alpha(G_{\mathcal{M}}) \end{aligned} \tag{4.4}$$

MIP formulation

Just like model (2.1), the following MIP-formulation (4.5) uses binary variables $Y_{i,j}$ for each arc $(i, j) \in A$ which indicate whether customer j is visited directly after customer i by the vehicle. As this formulation allows arbitrary customer sequences, it introduces further variables τ_i and c_i for all $i \in N$ to explicitly model delivery times at customers (τ_i) and production finishing times (c_i). This facilitates the formulation of time window constraints in condition (4.5b). Note that the summation terms on either side of this inequality are the same and that their value is equal to 1 if and only if customer i is supplied (resp. selected). If a customer i is not selected, the summation value is 0, thus enforcing $\tau_i = 0$.

$$\text{Maximize} \qquad P = \sum_{(i,j) \in A} p_i Y_{i,j} \tag{4.5a}$$

subject to

The time window constraints:

$$a_i \sum_{\{j|(i,j) \in A\}} Y_{i,j} \leq \tau_i \leq b_i \sum_{\{j|(i,j) \in A\}} Y_{i,j} \qquad (i \in C) \tag{4.5b}$$

The variable-coupling constraints (with M_τ and M_c being sufficiently large constants):

$$\tau_i + t_{i,j} - \tau_j \leq M_\tau(1 - Y_{i,j}) \qquad ((i,j) \in A) \qquad (4.5c)$$

$$c_i + p_j - c_j \leq M_c(1 - Y_{i,j}) \qquad ((i,j) \in A) \qquad (4.5d)$$

$$c_{n+1} \leq \tau_0 \qquad (4.5e)$$

The lifespan constraints:

$$\tau_i - c_i \leq B \qquad (i \in C) \qquad (4.5f)$$

The flow conservation constraints:

$$\sum_{\{h|(h,i)\in A\}} Y_{h,i} = \sum_{\{j|(i,j)\in A\}} Y_{i,j} \qquad (i \in C) \qquad (4.5g)$$

$$\sum_{j=1}^{n+1} Y_{0,j} = \sum_{i=0}^{n} Y_{i,n+1} = 1 \qquad (4.5h)$$

Using the variables:

$$Y_{i,j} \in \{0,1\} \qquad ((i,j) \in A) \qquad (4.5i)$$

$$\tau_i \geq 0 \qquad (i \in N) \qquad (4.5j)$$

$$c_i \geq 0 \qquad (i \in N) \qquad (4.5k)$$

The above mentioned variables are coupled by constraints (4.5c), (4.5d) and (4.5e). Inequalities (4.5c) state that the earliest delivery time at customer j is given by the delivery time at customer i plus the travel time $t_{i,j}$ if i is the immediate predecessor of j in the delivery sequence. Analogously, inequalities (4.5d) ensure that the production finishing time for j must be at least $c_i + p_j$ if j follows i directly. The values of the constants M_τ and M_c should be chosen as small as possible. Note, that inequality (4.5c) implies that $M_\tau \geq \max_{(i,j)\in A}\{\tau_i - \tau_j + t_{i,j}\}$. Since $\tau_i \leq b_i$ for all $i \in C$ in any feasible solution, using the definition

$$M_\tau := \max_{i\in C}\{b_i\} + \max_{(i,j)\in A}\{t_{i,j}\}$$

satisfies this requirement. Analogously, (4.5d) implies that $M_c \geq \max_{(i,j)\in A}\{c_i - c_j + p_j\}$. Since the difference $c_i - c_j$ between two finishing times is always at most the sum of all processing times, it is acceptable to set

$$M_c := \sum_{i\in C} p_i + \max_{i\in C}\{p_i\}.$$

The earliest time for the vehicle to leave the depot equals the production finishing time of the last order, which is guaranteed by (4.5e). By requiring the delivery time and the production finishing time to differ at most by the lifespan B for each customer $i \in C$ (4.5f), this model respects the lifespan constraints. Finally, restrictions (4.5g) and (4.5h) make sure that each customer is visited at most once and that the vehicle leaves and returns to the depot exactly once.

In this model, orders are always delivered in the same sequence as they are produced. However, it is straightforward to allow different sequences for production and deliveries by introducing further variables $X_{i,j} \in \{0,1\}$ for each $(i,j) \in A$ that represent the production sequence in the same manner as variables $Y_{i,j}$ do for the delivery sequence. In this case, variable $Y_{i,j}$ must be substituted by $X_{i,j}$ in restriction (4.5d), obtaining (4.5d'). Moreover, further flow conservation constraints (see (4.5l) and (4.5m) below) must be added to the model. Since an order i should be delivered if and only if it is also selected for production, variables X and Y must be coupled by constraints (4.5n).

Coupling of variables c and X in the production sequence:

$$c_i + p_j - c_j \leq M_c(1 - X_{i,j}) \qquad ((i,j) \in A) \qquad (4.5\text{d}')$$

The flow conservation constraints (for the production sequence):

$$\sum_{\{h|(h,i)\in A\}} X_{h,i} = \sum_{\{j|(i,j)\in A\}} X_{i,j} \qquad (i \in C) \qquad (4.5\text{l})$$

$$\sum_{j=1}^{n+1} X_{0,j} = \sum_{i=0}^{n} X_{i,n+1} = 1 \qquad (4.5\text{m})$$

Coupling of variables X and Y: Deliver an order if and only if it is produced:

$$\sum_{\{j|(i,j)\in A\}} Y_{i,j} - \sum_{\{j|(i,j)\in A\}} X_{i,j} = 0 \qquad (i \in C) \qquad (4.5\text{n})$$

4.2.2 Local search based on decomposition

In order to solve the SVESP we propose the following heuristic approach that represents an application of the tabu search metaheuristic. The idea behind this method is to decompose the SVESP into its two main parts: choosing a customer sequence for

production and deliveries as well as selecting a best possible subset for a given customer sequence.

These problems are addressed sequentially by first selecting a permutation π of all potential customers and then solving the customer selection problem by means of the algorithms presented in Chapter 2 for the fixed sequences problem. The latter algorithms are thus essentially used as a black box. Small instances allow using an exact algorithm like branch and bound for this inner step, whereas large instances require heuristic methods like TABUSEARCHFORSELECTIONS (see Algorithm 3.2 on p. 40) to obtain reasonable runtimes.

Our implementation alters the customer sequence in a local search fashion, i.e. by moving from the current sequence to another related sequence in a specified neighborhood. The most widely used neighborhoods for sequences (permutations) are defined by the following operations (see Brucker and Knust, 2006).

$\mathcal{N}_{api}$: Adjacent pairwise interchange

$\mathcal{N}_{swap}$: Swapping two arbitrary elements of the sequence

$\mathcal{N}_{shift}$: Relocating a customer within the sequence

The $\mathcal{N}_{api}$ neighborhood only allows a move from a sequence π to another sequence π' if π' can be obtained from π by interchanging a pair of adjacent elements in π. Since for $\pi = (\pi_1, \ldots, \pi_n)$ there are $n-1$ such interchanges, the neighborhood of π can be explored in $\mathcal{O}(n)$ time. The second neighborhood $\mathcal{N}_{swap}$ generalizes $\mathcal{N}_{api}$ in such a way that two arbitrary elements $1 \leq i < j \leq n$ can be interchanged. Accordingly, the $\mathcal{N}_{swap}$ neighborhood can be explored in $\mathcal{O}(n^2)$ time for a given sequence of n customers, because there are $\binom{n}{2} = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ ways to select a subset of two customers. The third neighborhood $\mathcal{N}_{shift}$ allows relocating a given customer i within the sequence, i.e. customer i is moved to another position and all customers between the previous and the new position of i are shifted forwards or backwards, respectively.

The neighborhood $\mathcal{N}_{shift}$ can also be explored in quadratic runtime, since a shift-operation is fully determined by choosing two indices $i \neq j$ which specify that π_i is moved to position j in π. Thus, there are $n(n-1)-(n-1) = (n-1)^2 = \mathcal{O}(n^2)$ possible shifts, because each of the n items can be relocated to $n-1$ other positions and the $n-1$ shifts resulting in adjacent swaps must not be counted twice.

In the following we will discuss our local search implementations using the $\mathcal{N}_{api}$ and $\mathcal{N}_{swap}$ neighborhoods. We do not explicitly cover variants using the $\mathcal{N}_{shift}$ neighborhood because they may be treated in a similar manner. However, different tabu criteria may apply.

Decomposition Tabu Search using the $\mathcal{N}_{api}$ neighborhood

Algorithm 4.1 Decomposition TS for the SVESP ($\mathcal{N}_{api}$ neighborhood)

```
 1: procedure DecompositionTS-Napi(S)
 2:     Set π ← S = (1,...,n) and σ_π ← SearchForBestSelection(π)
 3:     Initialize π* ← π, σ* ← σ_π, v* ← Value(σ_π)
 4:     Initialize tabu list TL ← ∅
 5:     repeat
 6:         Set bestSwap = nil, bestValue = −∞, bestSelection = ∅
 7:         for all non-tabu swap-moves (π_i, π_{i+1}) ∉ TL do
 8:             Execute swap π_i ⟷ π_{i+1} on π
 9:             Calculate selection σ_π ← SearchForBestSelection(π)
10:             p ← Value(σ_π)
11:             if p > bestValue then
12:                 bestSwap ← (π_i, π_{i+1}), bestValue ← p, bestSelection ← σ_π
13:             Undo last swap π_i ⟷ π_{i+1}
14:         Execute bestSwap on sequence π
15:         Update tabu list TL ← TL ∪ {bestSwap}
16:         if bestValue > v* then
17:             v* ← bestValue, π* ← π, σ* ← bestSelection
18:     until a stopping criterion is satisfied
19:     return (π*, σ*)
```

Our decomposition-based tabu search called DecompositionTS-Napi uses the $\mathcal{N}_{api}$ neighborhood and is presented as Algorithm 4.1. It it composed of two nested loops, where the outer (repeat-) loop runs until a certain stopping criterion is satisfied. This criterion might, for example, be specified as an execution time limit or a maximum number of iterations. Another common stopping criterion consists in finishing the tabu search if no improvement of the best known solution could be obtained during

a pre-specified number of iterations. The inner (for-) loop (lines 7–13) evaluates all possible (non-tabu) neighborhood-moves, searching for a pair (π_i, π_{i+1}) of adjacent customers in π to be swapped so that the total satisfied demand, which is computed by the search for selections, for the newly obtained sequence is maximum. Note that SEARCHFORBESTSELECTION in lines 2 and 9 should be regarded as a placeholder, either standing for an exact or a heuristic method, as explained above. In each iteration of the outer loop, a best swap is executed and the corresponding pair of customers (stored as $bestSwap$) is included in the tabu list TL. The tabu status of this pair now specifies that the search is not allowed to reverse this swap in the following $|TL|$ iterations of the repeat-loop. In our implementation we use a fixed-size tabu list, where only the latest $|TL|$ insertions are stored.

A returned solution consists of a pair (π^*, σ^*), such that π^* is a permutation of all customers and σ^* is a corresponding customer selection, typically represented by a bit vector. The objective function value of such a solution is computed in Algorithm 4.1 in lines 3 and 10. Denoting the set of feasibly supplied customers by $F(\sigma_\pi) := \{i \in C \mid \Delta_i = l_i = 0\}$, we define $\text{VALUE}(\sigma_\pi) := \sum_{i \in F(\sigma_\pi)} p_i$. This means that we sum up the demands of all feasibly supplied customers, i.e. the selected customers $i \in \sigma_\pi$ who receive their deliveries without tardiness ($\Delta_i = 0$) as well as without lifespan violation ($l_i = 0$).

Decomposition TS using the $\mathcal{N}_{swap}$ neighborhood

The presented decomposition approach for the SVESP can also be applied to the case, where the $\mathcal{N}_{swap}$ neighborhood is used. Nevertheless, it appears necessary to alter the tabu criterion because the simple criterion of forbidding the reversal of a number of recent swaps does not prevent the search from cycling. More precisely, it is possible to construct a sequence of swaps that can be carried out over and over without eventually changing the customer sequence. The following example clarifies this fact. Consider a (sub-) sequence of four customers (i, j, k, l):

$$
\begin{array}{lll}
 & (i,j,k,l) \mid i \leftrightarrow j & (1) \\
\hookrightarrow & (j,i,k,l) \mid k \leftrightarrow l & (2) \\
\hookrightarrow & (j,i,l,k) \mid i \leftrightarrow l & (3) \\
\hookrightarrow & (j,l,i,k) \mid j \leftrightarrow k & (4) \\
\hookrightarrow & (k,l,i,j) \mid k \leftrightarrow i & (5) \\
\hookrightarrow & (i,l,k,j) \mid l \leftrightarrow j & (6) \\
\hookrightarrow & (i,j,k,l) & (7)
\end{array}
$$

Executing these six different swaps obviously restores the initial customer sequence. This means that the employed tabu criterion cannot prohibit the given cycle. Furthermore, if the length of the tabu list is at most 5 in the above example, the indicated cycle may be repeated until the iteration limit is reached, preventing the search from exploring other feasible solutions.

A more appropriate tabu criterion for a swap neighborhood on permutations has been proposed by Glover and Laguna (1997), who develop the following main idea: If two customers π_i and π_j, $1 \leq i < j \leq n$, are swapped in the customer sequence π during iteration $Iter$, this results in the following update:

$$FT[\pi_i] \leftarrow Iter \quad \text{and} \quad BT[\pi_j] \leftarrow Iter$$

The arrays FT and BT record the iteration numbers in which π_i and π_j are set tabu. Their names are acronyms for *Forward Tabu* and *Backward Tabu*, respectively, which means that if the entry of customer π_i in the array FT (BT) is updated in iteration $Iter$, the tabu search is not allowed to move π_i to an earlier (a later) position in π for the subsequent $Tenure$ iterations, where $Tenure$ represents a pre-specified parameter of the search. A tabu classification using these definitions is given as follows: The candidate move of swapping customers π_i and π_j, where $i < j$, is considered tabu if the following condition is satisfied:

$$Iter \leq \max\left\{BT[\pi_i],\ FT[\pi_j]\right\} + Tenure \tag{4.6}$$

This means that we compute the more recent of the two times, when customer π_i was moved backward or π_j was moved forward, respectively, and finally add the $Tenure$ parameter to determine the iteration until that the swap $\pi_i \longleftrightarrow \pi_j$ stays tabu.

In fact, this tabu criterion also succeeds in preventing the cycling process in the above mentioned example for the sequence (i, j, k, l). If we assume that $Tenure = 5$ and we consider step (4) in the example, where the swap $j \longleftrightarrow k$ is supposed to be executed, then the new criterion makes this swap tabu, because j has most recently been moved forward in step (1) and thus $BT[j] = 1$. Furthermore, customer k has most recently been moved backward in step (2), thus $FT[k] = 2$. According to inequality (4.6), the swap $j \longleftrightarrow k$ will be tabu until time $\max\{BT[j], FT[k]\} + Tenure = 2 + 5 = 7$, which would prevent its execution in iteration (4) of the above example.

Procedure DECOMPOSITIONTS-NSWAP in Algorithm 4.2 summarizes the described tabu search in pseudocode. In step 4, all entries of the arrays FT and BT are initialized to $-\infty$, ensuring that none of the possible swaps is tabu. In the main iteration for-loop, the tabu status of each possible swap (π_i, π_j) is checked by using (the inverse of) the condition in (4.6). If the swap is not tabu, it is tentatively carried out and a customer

Algorithm 4.2 Decomposition TS for the SVESP ($\mathcal{N}_{swap}$ neighborhood)

```
procedure DecompositionTS-Nswap(S)
    Set π ← S = (1, ..., n) and σ_π ← SearchForBestSelection(π)
    Initialize π* ← π, σ* ← σ_π, v* ← Value(σ_π)
    Initialize FT[i] ← BT[i] ← −∞ for each i = 1, ..., n
    for Iter ← 1 to MaxIter do
        Set bestValue = −∞, bestSelection = ∅
        for all pairs (i, j) with 1 ≤ i < j ≤ n do
            if Iter > max{BT[π_i], FT[π_j]} + Tenure then
                Execute swap π_i ⟷ π_j on π
                Calculate selection σ_π ← SearchForBestSelection(π)
                p ← Value(σ_π)
                if p > bestValue then
                    (π*_i, π*_j) ← (π_i, π_j), bestValue ← p, bestSelection ← σ_π
                Undo last swap π_i ⟷ π_{i+1}
        Execute swap π*_i ⟷ π*_j on sequence π
        Set FT[π_i] ← Iter and BT[π_j] ← Iter
        if bestValue > v* then
            v* ← bestValue, π* ← π, σ* ← bestSelection
    return (π*, σ*)
```

selection σ_π corresponding to the altered sequence is computed. During each iteration, the procedure searches for the best possible swap, i.e. the swap that yields the largest total satisfied demand. Note that it is possible that even the best swap may not result in an improvement of the current solution. In this case, a deteriorating move is carried out, which is sometimes necessary to escape from local optima and to diversify the search.

4.2.3 Local search for selection lists

Arguably, the decomposition approach for solving the SVESP presented before suffers from the lack of coordination between inner and outer search algorithms. On the one hand, the procedure responsible for the inner search step, SearchForBestSelection, can be substituted by either an exact or a heuristic method, depending on the considered problem sizes and/or computation time requirements. On the other hand, due to the "black-box" approach the only feedback from a run of SearchForBestSelection witnessed by the outer search consists in the computed selection and its associated objective function value. This only partially allows conclusions about the contribution of a specific neighborhood-step in the outer search. Furthermore, for many problem

instances the outer search algorithm needs to evaluate swaps even for those customers who will never be part of an optimal solution.

A method to overcome these difficulties is based on a different representation of solutions that we call **selection lists**. A selection list σ for a set of customers S consists of an ordered subset of S. The heuristic algorithm we propose below is based on this representation and it will always divide the set S into a current selection list σ and the set of unselected customers $\bar{\sigma} := S \setminus \sigma$ which may or may not be considered as sorted, for example according to non-increasing demands or a time-related flexibility criterion. Maintaining $\bar{\sigma}$ as a sorted set can accelerate the process of choosing a suitable customer to be selected in the next local search step.

Algorithm 4.3 is again based on the tabu search paradigm; it is composed of a two-phase heuristic that is split into a starting phase (GenerateStartingSolution) and an improvement phase (SelectionListTS). The starting phase follows a greedy approach as it iteratively tries to include each customer at the respective best position. Insertion attempts are conducted according to a preceded sorting of the customers, where the sorting criterion consists of a sum of demand and width of time window (flexibility of delivery), weighted by a factor $0 \leq \alpha \leq 1$ (cf. line 23). In our empirical evaluation, values for α between 0.6 and 0.8 led to the best results, that is, stronger emphasis should be put on increasing the satisfied demand, but considering the flexibility criterion helps in doing so. Respecting widths of time windows for the sorting is motivated by the expectation that the generated schedules will stay flexible for a longer period, as customers are added to the selection list. Thus, in the end, probably more customers can be feasibly included in the starting solution.

Procedure SelectionListTS triggers the execution of the starting heuristic and tries to improve the obtained solution via tabu search. The tabu list TL stores customers who are currently not allowed to be considered for insertion. In each iteration, an unselected customer $c \in \bar{\sigma}$ that is not contained in TL is selected for insertion (cf. line 7). As more and more customers become selected, it usually happens at some point that such a customer c cannot be feasibly inserted because the time window or lifespan constraints for c (or other customers in σ) cannot be satisfied. In this case, SelectionListTS computes the set $\mathcal{R}_c \subseteq \sigma$ (cf. line 9) of customers such that the removal of a single customer $s \in \mathcal{R}_c$ from σ allows the feasible insertion of c at some position in $\sigma \setminus \{s\}$. If the set $\mathcal{R}_c$ is empty, no further attempt is made to insert c and we mark c as tabu. Otherwise, we exclude a customer $r \in \mathcal{R}_c$ with smallest demand p_r from the selection, mark it as tabu and insert c at the best possible position into σ. This position is determined by computing the objective function value Value(σ') of each (shift-) feasible selection list σ' that is obtainable by inserting c at some position into σ. However, sometimes the value of the objective function does not directly

Algorithm 4.3 Tabu search for selection lists

```
procedure SelectionListTS(S)
    σ ← GenerateStartingSolution(S)
    Initialize σ* ← σ and v* ← Value(σ)
    Initialize tabu list TL ← ∅
    repeat
        if σ̄ = ∅ then return σ
        Choose a non-tabu customer c ∈ σ̄ to be inserted
        if insertion of c into σ is (shift-) infeasible for each position then
            Set R_c ← { s ∈ σ | c can be (shift-) feasibly inserted into σ \ {s} }
            if R_c = ∅ then
                Set TL ← TL ∪ {c}
            else
                Find r ∈ R_c having smallest demand p_r
                Remove r from σ and set TL ← TL ∪ {r}
                Insert c at best possible position into σ
        else
            Insert c at best possible position into σ
        if Value(σ) > v* then σ* ← σ and v* ← Value(σ)
    until a stopping criterion is satisfied
    return σ*

procedure GenerateStartingSolution(S)
    σ ← ∅
    Sort S according to non-increasing values αp_c + (1 − α)(b_c − a_c), c ∈ S
    for all customers c ∈ S in sorted order do
        if c can be (shift-) feasibly inserted into σ then
            Insert c at best position (best objective value) into σ
    return σ
```

depend on the ordering of the selected customers in the delivery sequence, like for the "total satisfied demand" objective, where each feasible insertion position yields the same value. In this case, it is sufficient to find *any* feasible insertion position, which may be exploited to improve runtimes.

4.3 Differing production and distribution sequences

In Example 2 (p. 46) we have demonstrated that in the presence of limited product lifespans there are problem instances which admit schedules with better objective

function values if production and distribution sequences σ^P, σ^D are allowed to differ. In the following, we will discuss this scenario, which introduces a second sequencing subproblem, namely the separate (but still integrated) determination of a production sequence σ^P and distribution sequence σ^D such that both sequences are permutations of each other. We will refer to this setting as the **single vehicle, differing sequences problem (SVDSP)**.

By definition, the SVDSP represents a generalization of the SVESP and the necessary changes and additions to the SVESP-MIP-model have already been discussed before in Section 4.2.1 (p. 53). Our computational experiments have confirmed that solving the resulting MIP-model for the SVDSP directly by standard MIP solvers is only tractable for small problem instances that contain at most 20 customers. This limit actually depends on the instance characteristics, such as the width of time windows or the length of the product lifespan compared to travel times. However, problem instances of more than 30 customers proved to be intractable in the majority of cases, which leads to the conclusion that heuristics need to be employed for coping with practical problem sizes.

The SVDSP draws much of its complexity from its constituent subproblems. Assume that we fix a production sequence σ^P and a production start time c_0. If minimizing the distribution time or the total distance traveled is at least part of the objective function, then the remaining problem corresponds to a traveling salesman problem with time windows (TSPTW), where the given customer time windows can possibly be narrowed by taking the expiration dates of the individual orders into account. These dates are given by the fixation of σ^P, which also determines the distribution start time $\tau_0 = c_0 + \sum_{i \in \sigma^P} p_i$. The TSPTW is known to be NP-hard in the strong sense (Garey and Johnson, 1979).

As a basis for developing heuristics, the representation of solutions for the SVDSP by pairs of lists (σ^P, σ^D) is reasonable because such a pair completely determines the production and delivery schedules, except for the start time of the production phase. This is due to the fact that the production is conducted by a single machine without idle times between different orders, and the distribution schedule is obtained as an earliest start schedule, beginning at the end of the production phase. We can account for possible shifts of the production start by applying the delaying criteria derived in Section 3.3.

4.3.1 Starting solutions

Determining the distribution sequence σ^D typically constitutes the more difficult part in solving the SVDSP because its feasibility is subject to time window constraints.

Note that deciding whether there is a feasible solution for an instance of the traveling salesman problem with time windows is already a strongly $\mathcal{NP}$-complete problem (Savelsbergh, 1985). On the other hand, the production sequence σ^P merely needs to guarantee that the product lifespans last long enough to allow feasible deliveries within the customer time windows. According to this requirement, using a starting solution with equal sequences (i.e. $\sigma^P = \sigma^D$) seems at least plausible because in such a schedule the deliveries take place in the same order as the products expire. Hence, a solution to the SVESP may often be a good starting point for the SVDSP, although one might argue that such a starting solution is too much biased towards equal sequences.

Insertion heuristic

In the following we describe an alternative method to obtain a starting solution from scratch. The general idea is to start from an empty customer selection, sort the customers in S according to non-increasing demands and iteratively try to insert customers respecting the sorted sequence. The insertion step itself is performed using a method which was introduced by Solomon (1987) and which is an essential part of Solomon's well-known I_1-heuristic for the vehicle routing problem with time windows (VRPTW). In a nutshell, Solomon's insertion heuristic iteratively adds a customer that has the lowest cost of insertion of all remaining customers, at the corresponding best possible position in the current (partial) sequence. However, our starting heuristic does not select the next customer to be inserted according to insertion costs but rather according to a preceding sorting of the customers that respects the individual demands. We merely make use of Solomon's insertion criterion for finding the best possible position for a given customer. We will now briefly describe this criterion.

Assume that customer r is chosen to be inserted into the current (delivery) sequence $\sigma^D = (0 = s_0, s_1, \ldots, s_k, s_{k+1} = n+1)$. Note that we have exceptionally included the depot nodes 0 and $n+1$ in σ^D to simplify the following description and avoid special cases for inserting r at the beginning or at the end. The cost of inserting r between customers s_i and s_{i+1}, $i = 0, \ldots, k$, is defined in the following equation as the convex combination of two components (with parameter $0 \leq \alpha \leq 1$)

$$C(i,r) := \alpha C_1(i,r) + (1-\alpha) C_2(i,r)\,,$$

where the first component

$$C_1(i,r) := t_{s_i,r} + t_{r,s_{i+1}} - \mu t_{s_i,s_{i+1}}$$

represents the parameterized costs ($\mu \geq 0$) for the additional transport time and the second component

$$C_2(i,r) := \tau'_{s_{i+1}} - \tau_{s_{i+1}}$$

accounts for the delay in the delivery time at the directly following customer s_{i+1}, where $\tau'_{s_{i+1}}$ indicates the delivery time *after* the insertion and $\tau_{s_{i+1}}$ *before* the insertion. If this operation causes a delivery tardiness (time window infeasibility) for at least one of the already included customers, we set $C_2(i,r) := \infty$ in order to penalize this insertion position. Our heuristic chooses a position $i^* \in \{0,\ldots,k\}$ such that $C(i^*,r)$ is as small as possible over all insertion positions. If none of these is feasible, i.e. if $C_2(i,r) = \infty$ for all $0 \leq i \leq k$, we skip customer r in the sorted customer sequence and try to include the following customer if possible.

Algorithm 4.4 Computing a starting solution for the SVDSP

1: **procedure** STARTINGSOLUTIONSVDSP(S)
2: Sort S according to non-increasing demands
3: Initialize $(\sigma^P, \sigma^D) \leftarrow (\emptyset, \emptyset)$
4: **for all** customers r in sorted sequence S **do**
5: Append r to production sequence σ^P
6: Insert r into σ^D via Solomon's insertion criterion
7: **if** (σ^P, σ^D) is infeasible due to time window restrictions **then**
8: Remove r from σ^P and σ^D (and try next customer)
9: **else if** (σ^P, σ^D) is infeasible due to lifespan violations **then**
10: Let $\sigma^{late} \leftarrow \left\{ s \in \sigma^P \mid l_s > 0 \right\}$
11: Sort σ^{late} according to non-increasing demands
12: Move customers in σ^{late} (in sorted order) to the end of σ^P
13: **if** (σ^P, σ^D) is still infeasible **then**
14: Move customers in σ^{late} back to their previous positions
15: Remove r from σ^P and σ^D (and try next customer)
16: **return** (σ^P, σ^D)

Algorithm 4.4 provides a pseudocode overview of the insertion method discussed before. Since Solomon's I_1-heuristic is not designed to respect lifespan constraints, procedure STARTINGSOLUTIONSVDSP contains a corresponding check in line 9 that is only carried out if the current solution (σ^P, σ^D) is at least time window feasible. If this check fails and a lifespan infeasibility is discovered (while time window constraints are obeyed), we try a simple repair heuristic which attempts to remedy the lifespan violations by reordering the production sequence in such a way that late orders (for customers $s \in \sigma^P$ with positive tardiness $l_s > 0$) are moved to the end of the production process. Sorting these customers according to non-increasing demands,

which is done in line 11, postpones their production completion times as far as possible. Consequently, this makes the goods expire as late as possible. If the repair heuristic is unsuccessful, the respective customers are moved back to their previous positions and customer r is skipped.

Our adaption of Solomon's insertion method depends on the parameters α and μ, where α controls the precedence of minimizing the additional travel time vs. the shift in the delivery time at the succeeding customer, and μ specifies a "weight" for the removed edge (s_i, s_{i+1}) in the travel graph. Large values for μ put more emphasis on removing long direct edges in favor of the two inserted edges for the additional visit. To obtain the well-known savings criterion for the insertion costs (Clarke and Wright, 1964), the value $\mu = 1$ should be chosen.

4.3.2 Fast feasibility checks for insertions

Inserting a currently unselected customer into a given production/distribution plan (σ^P, σ^D) is a frequent operation in many conceivable heuristics for the SVDSP and similar problems. For this reason, it is important that the feasibility of the resulting plan can be checked efficiently for each insertion step.

For the SVDSP this validation is far from trivial, because inserting a customer is not restricted to cause *local* changes in the current production/distribution plan, but it exerts a *global* influence on it. Since the feasibility of the distribution sequence σ^D is subject to time window constraints, we consider inserting a customer r into σ^D first, whereas finding a position for r in σ^P is a subordinate goal.

The insertion implies the following changes to the current plan:

- The production phase is prolonged by p_r time units. In the worst case, r is simply appended to σ^P such that its production time consumes a fraction of the lifespan of *all* previously produced orders.
- The earliest start time for the distribution phase is delayed by p_r time units, which may incur a delivery tardiness at some customers. In general, the delivery times for *all* customers must be re-calculated.
- The insertion of r into σ^D generally delays the earliest arrival times at the customers behind the insertion position. This must be in accordance with the allowed maximum transport delay (see Def. 1, p. 30) of the subsequent customer.

An obvious (but inefficient) way of checking the feasibility of inserting r at a particular position in σ^D consists in computing the earliest start schedule (ESS) for the extended plan $(\sigma^P_{+r}, \sigma^D_{+r})$ and validating the lifespan and time window constraints. This check

can be implemented to run in $\mathcal{O}(k)$ time, where k denotes the number of previously selected customers. We finally end up with an $\mathcal{O}(k^2)$ runtime algorithm if insertion is tested at each possible position. However, we can do better by making use of the values for the maximum transport delay $\delta^T_{s_i}$, $i = 1, \ldots, k$, as mentioned above. Recall that $\delta^T_{s_i}$ represents the maximum allowed delay in the arrival time at customer s_i such that the lifespan and time window restrictions for s_i and all subsequent customers $s_{i+1}, \ldots, s_k$ are still obeyed.

In the following we will present an $\mathcal{O}(k)$ runtime algorithm which

- checks for a given unselected customer r and a production/distribution schedule (σ^P, σ^D), whether r can be feasibly inserted and
- finds a corresponding position for r in σ^D.

First, we assume that the current production/distribution schedule (σ^P, σ^D) is given by production completion times c_{s_i} and delivery times τ_{s_i} for all $i = 1, \ldots, k$, and that the maximum transport delay $\delta^T_{\sigma^D}$ has been calculated according to Definition 1, which can be done in $\mathcal{O}(k)$ runtime.

We initially check whether we can delay the distribution phase by the necessary production time for r, i.e. we must have $p_r \leq \delta^T_{\sigma^D}$. If this condition is violated, r cannot be inserted. Otherwise, we calculate new delivery times τ'_{s_i} and maximum transport delays $\delta^{T\prime}_{s_i}$ for all $i = 1, \ldots, k$ according to the delayed distribution start caused by the additional production time. This can be achieved by computing the new ESS and a subsequent backwards-calculation for the delays as specified by Definition 1. Note that the described steps need to be carried out only once for the insertion of customer r, and not for each position where an insertion is tested. The aggregate runtime of those steps is $\mathcal{O}(k)$.

After that, we need to test insertion of r potentially at every position $i = 0, \ldots, k$ in σ^D unless we can derive precedence relations from the customer time windows such that some positions can be skipped. By inserting customer r at position i in σ^D we mean inserting r between customers s_i and s_{i+1}. Figure 4.2 (a) depicts the situation after the recalculation of delivery times and max. transport delays, but before the insertion of r, whereas part (b) shows the changes after the insertion. Note that only the regarded section of the distribution sequence is presented in the figure.

The feasibility test for insertion of r at position i proceeds as follows: We set $\tau''_{s_i} := \tau'_{s_i}$ (the delivery at s_i is unchanged) and calculate the delivery time for r:

$$\tau''_r := \max\left\{a_r,\ \tau''_{s_i} + t_{s_i,r}\right\}$$

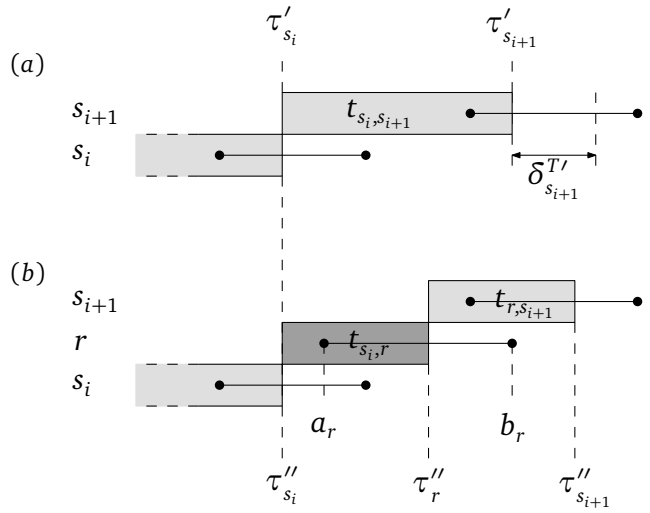


Figure 4.2: Inserting customer r between s_i and s_{i+1} in the distribution sequence

After that, we check whether the following conditions are satisfied:

$$\tau''_r \leq \min\{b_r,\ c_r + B\} \tag{4.7a}$$

$$\tau''_r + t_{r,s_{i+1}} \leq \tau'_{s_{i+1}} + \delta^{T\prime}_{s_{i+1}} \tag{4.7b}$$

Inequality (4.7a) tests whether r is delivered within its time window and the product lifespan (beginning at c_r and not shown in the figure), whereas (4.7b) ensures that the following customer s_{i+1} can be served before its maximum transport delay limit. If both conditions hold, then i is a candidate position for insertion of r. Otherwise, the insertion would either violate time window or lifespan conditions for r itself or it would incur a delay in the delivery schedule, causing the service at at least one of the succeeding customers to become infeasible.

A single test for a fixed insertion position i takes constant time. Together with the costs of recalculating delivery times and delays, testing the insertion of r at each possible position in σ^D takes $\mathcal{O}(k)$ time in total, which improves upon the $\mathcal{O}(k^2)$ bound set by the "obvious" method mentioned above.

4.3.3 Improvement heuristic

In order to improve a given starting solution for the SVDSP, we have developed a metaheuristic method that is based on **Iterated Local Search (ILS)**. ILS was introduced by Lourenço et al. (2003) and it captures the idea of iteratively building a sequence of solutions that are computed by an embedded heuristic, which is often treated as a

"black box". However, the latter is no requirement. The approach goes beyond merely running the embedded heuristic from random starting solutions, which would deny any knowledge obtained from previous search trials. Instead, ILS follows a single chain of locally optimal solutions with respect to the embedded heuristic, i.e. the search for better solutions is performed in a reduced search space.

Algorithm 4.5 Generic Iterated Local Search (ILS)

```
1: procedure GenericILS
2:     s_0 ← GenerateStartingSolution()
3:     s* ← EmbeddedLSHeuristic(s_0)
4:     repeat
5:         s' ← Shake(s*)
6:         s'' ← EmbeddedLSHeuristic(s')
7:         s* ← AcceptOrNot(s*, s'')
8:     until a stopping criterion is satisfied
9:     return s*
```

A very generic view on ILS is shown in Algorithm 4.5. After generating a starting solution, the embedded local search heuristic is applied to it, thus defining the incumbent solution s^*. In each iteration of the repeat-loop, a perturbation operation called SHAKE is applied to the current solution, which leads to an intermediate state s' that is not locally optimal in general. The shaking operation provides ILS with the ability to escape from local optima, typically by destroying a specified fraction of the current solution (e.g. by removing certain customers from a selection). After the shaking process, the corresponding local optimum is found by executing local search on s', resulting in a solution s''. Finally, the third step in the loop comprises the decision whether s'' is accepted as the new current solution or not.

On the one hand, if the search only accepts better solutions, a first-improvement-type descent heuristic is obtained. On the other hand, accepting each generated solution regardless of its costs yields a random walk behavior, which is not necessarily unfavorable, when, for example, the solution space for a given problem is known to be frequently "peaked" around local optima such that randomization helps to escape from them.

Lourenço et al. (2003) point out that ILS generally works best if the shaking operation on s^* can not be easily undone by the embedded local search heuristic and if the ACCEPTORNOT procedure is not too biased towards better solutions. The main goal of ILS consists in thoroughly sampling elements from the set of locally optimal solutions with respect to the embedded LS heuristic.

Application of ILS to the SVDSP

Vansteenwegen et al. (2009b,c) describe an application of ILS in the context of tourist trip planning, where the problem is modeled as an orienteering problem with time windows (OPTW). In the OPTW, one has to find a single tour through a given set of places, where each place is associated with a profit value and a visiting time window, and the route has a duration limit. Furthermore, each place can be visited at most once, and by visiting a place the associated profit is collected. The objective is to find a tour through a subset of places such that the total collected profit is maximized subject to time window and tour duration constraints.

Computational results of Vansteenwegen et al. (2009c) have established that ILS performs well on their set of problem instances, even when considering the extended team orienteering problem with time windows (TOPTW), where multiple tours have to be determined.

The OPTW exhibits many similarities with the single vehicle problem with differing sequences, e.g. the requirement of meeting time windows, the specification of profit values (demands correspond to revenues) at the places that can be visited, but most notably the underlying combinatorial problem structure. Obviously, both problems share the subproblems of simultaneously selecting a subset of vertices and determining an appropriate tour through the selected vertices. Depending on the considered objective function in the SVDSP, an optimal tour may be represented by a shortest tour among all tours in that subset according to total travel time or total duration, respectively. Viewed in this light, both the OPTW and the SVDSP can be regarded as combinations of the *Binary Knapsack Problem* and the *Traveling Salesman Problem with Time Windows*.

Due to the above-mentioned resemblance of the two problems and the good results obtained for the OPTW, we have chosen to address the SVDSP with an ILS-based heuristic, whose components are described in more detail in the following.

Initial Solution An initial solution s_0 for the iterated local search is obtained using the insertion heuristic presented in Section 4.3.1.

Embedded Heuristic (Local Search) The embedded heuristic tries to maximize the total satisfied demand by inserting additional customers one-by-one into the current tour. This process terminates if either all customers are inserted or if there is no unsupplied customer left that can be feasibly inserted. Actually, we use a method which is very similar to that for generating the initial solution (c.f. Algorithm 4.4). Instead of sorting the entire customer set S, we only sort

the set of currently unselected customers $S \setminus \sigma^P$ according to non-increasing demands and then attempt to insert them into the current production/distribution sequence (σ^P, σ^D). Furthermore, we also keep a record on the frequency with which each customer is successfully inserted. Beginning at a certain number of iterations (determined by a parameter), we also consider these frequencies in the sorting such that those unselected customers are preferred who are likely to be effectively integrated.

Note that, according to this choice of the embedded heuristic, the starting heuristic already covers steps 2 and 3 of the generic ILS in Algorithm 4.5, since the result of procedure StartingSolutionSVDSP is already a local optimum with respect to the embedded heuristic.

Shaking (Perturbation) The shaking step follows the primary goal to provide the search with the capability to escape from local optima. In our case, being able to escape from local optima means that customers must be removed from the current (locally optimal) solution (σ^P, σ^D) such that the embedded heuristic gets the chance to insert a possibly different set of unselected customers afterwards. More specifically, this is done in the following way: Shake deselects one or more customers according to two parameters called `ShakeLength` and `ShakePosition`. `ShakeLength` indicates how many consecutive customers in σ^D will be removed, whereas `ShakePosition` specifies the position in σ^D where the removal process begins. Of course, each deleted customer is also removed from the production sequence σ^P as well. This is done in a cyclic manner, i.e. when the last customer in σ^D is reached, the removal process is continued at the start. Clearly, the entire process entails a recalculation of delivery times (and dependent values) for the remaining customers in the sequence.

As a means of diversification and intensification of the search, both shaking parameters are altered during the iterations of the ILS. After each shaking step, `ShakePosition` is increased by the value of `ShakeLength` (modulo sequence length), while `ShakeLength` is increased by one. Each time a new best solution is encountered or when `ShakeLength` is at least half the number of selected customers, `ShakeLength` is reset to 1. Changing the parameters in this way makes it very likely that different customers are removed in each shaking step which assists in better exploring the search space.

Acceptance criterion Experiments by Vansteenwegen et al. (2009c) with different acceptance criteria applied to ILS for the TOPTW have confirmed that accepting each solution that is generated by the embedded insertion heuristic represents the most successful implementation. For this reason, our ILS method also accepts each heuristic solution, irrespective of its costs. Clearly, this leads to a kind

of random dynamics in the search process. Randomization, however, seems necessary since the solution space for the SVDSP appears quite “rough”, which means that small changes in the solution representation may quickly lead to infeasible solutions.

Chapter 5

Multiple tours and vehicles

A general assumption in the basic model presented in Chapter 2 is that the single transporter has sufficient capacity to serve all of the customers' demands on a single tour if that is possible at all with regard to time window or lifespan constraints.

In fact, it is more realistic to assign a fixed capacity to the transporter and permit multiple tours per workday. This allows the vehicle to leave the depot before all of the accepted orders have been produced, which in turn entails a reconsideration of the production scheduling because in this situation it may be advantageous – due to lifespan restrictions – to delay certain production steps, depending on the time at which the vehicle returns to the depot. A generalized production and distribution model that allows multiple tours for a single vehicle during the planning period is discussed in Section 5.1.2.

Another generalization consists in the scenario, where several transporters with possibly different capacities and/or travel speeds are responsible for the deliveries. In addition to solving the sequencing subproblems for the deliveries of each vehicle it is then necessary to assign each customer to a transporter (a process known as clustering) or, more generally, to a specific tour of one of the transporters in order to optimize a given objective function. For this model variant it is often appropriate to include penalty costs in the objective, which serve to minimize the number of vehicles needed for the deliveries and/or the total distance traveled by the selected set of vehicles.

This inclusion poses the difficulty of determining adequate cost factors for (unsatisfied) demands and vehicle costs in the objective function because these goals (min. number of vehicles vs. max. sat. demands) are contradictory. This is due to the fact that the delivery problem generally becomes easier to solve if there are sufficiently many vehicles available (with fixed costs left unconsidered) to serve all of the orders. In an

extreme case, each customer could be served by a dedicated transporter. Actually, this situation is not so unusual, as it is encountered, for example, in ready-mix concrete delivery. In most cases, however, using an extra vehicle involves additional fixed costs, implying that it is generally not reasonable to use as many vehicles as possible to deliver all orders on time. As a consequence, the goal is to find a reasonable tradeoff between the number of employed vehicles, their corresponding travel costs and satisfied demands.

In the next section we introduce an extended variant of the models for variable customer sequences presented in the last chapter, where multiple consecutive delivery tours are possible.

5.1 A model for the single vehicle, multiple tours, variable sequence problem (SVMTP)

In the following we generalize the problem presented in Section 2.1 by allowing a single vehicle (now having a fixed capacity q) to deliver orders on **multiple tours**, i.e. the vehicle may return to the depot and load further goods to be delivered on its next tour. If goods are loaded onto the vehicle at the depot, they are assumed w.l.o.g. to be delivered on the tour directly afterwards, that is, the vehicle will return empty.

We call this scenario the **single vehicle, multiple tours, variable sequence problem (SVMTP)**. Introducing vehicle capacities and intermediate depot returns essentially introduces concurrency into the scheduling problem because then it is often reasonable to produce goods for the next tour while the vehicle is still away from the depot, provided that keeping inventories until the start of the next vehicle tour is possible.

5.1.1 Parameters and decision variables

We assume that the tours are disjoint, which means that the vehicle never visits a customer more than once. This implies that partial deliveries of orders are not allowed. However, due to the time window constraints, possibly not all customers may be supplied, even in an optimal production/distribution plan. The setting is depicted in Figure 5.1. There, the delivery sequence is denoted by $\sigma^D = (\sigma_0^D, \ldots, \sigma_{i_k}^D)$, $k \geq 1$, where the intermediate depot returns for the k tours are marked by (dummy) customers $\sigma_{i_1}^D, \ldots, \sigma_{i_k}^D$. Using this notation, the j-th tour contains the (real) customers $\sigma_{i_{j-1}+1}^D, \ldots, \sigma_{i_j-1}^D$, where $j = 1, \ldots, k$ and $i_0 := 0$.

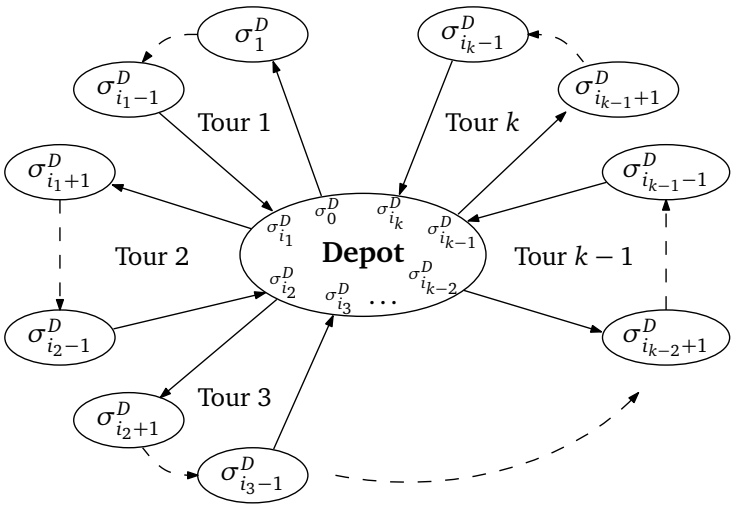


Figure 5.1: Multiple consecutive tours for a single vehicle

By $C := \{1, \ldots, n\}$ we will denote the **set of customers**. Furthermore, we will call $N := C \cup \{0, n+1\}$ the **set of nodes** and $A := \{(i, j) \mid 0 \leq i \leq n,\ 1 \leq j \leq n+1,\ i \neq j\}$, the **set of arcs** of the transport graph, where the depot (production facility) is referred to as node 0 or $n+1$, depending on whether it is the initial or terminal node of an arc. This means that we introduce arcs between each pair of nodes, but neither ingoing arcs for node 0 nor outgoing arcs for node $n+1$. It is possible to alter the definition of set A by removing arcs that cannot be part of a feasible solution to the scheduling problem. For example, if i and j are nodes of the transport graph satisfying $a_i + t_{i,j} > b_j$, then there is no way for the vehicle to visit i and immediately afterwards j in a feasible schedule. In this case, arc (i, j) can be removed from A. Similar preprocessing steps are discussed in Section 4.2.1 on page 50.

The formulation given below uses an ordered set of **possible tours** $K = \{1, \ldots, k\}$ for the vehicle. We assume w.l.o.g. that the tours are served in ascending order $1, 2, \ldots, k$, while we intentionally allow some of these tours to be empty; in the latter case the vehicle stays at the depot, modeled by traveling directly from node 0 to $n+1$ without incurring any costs. Hence, k represents the maximum number of allowed tours. Of course, an actual solution may not need as many as k tours. But the value of parameter k should be chosen carefully to keep the number of decision variables low while still permitting as many tours as needed for obtaining high quality solutions.

In contrast to the fixation of the delivery (and production) sequence in the basic model we allow **arbitrary delivery sequences**. However, for now we will restrict the

production sequence to be equal to the delivery sequence, just as we did in the model for the SVESP in Section 4.2.

The model presented here is related to the one used by Azi et al. (2007, 2010) for the Vehicle Routing Problem with Time Windows (VRPTW), where perishable goods are transported and each vehicle can perform several tours during its workday. Our model needs to reflect more restrictions than in the VRPTW due to the interdependence with the production phase and the lifespan constraints for the product, but the formulation of the distribution phase is very similar. Furthermore, not the entire set of customers must be served, but a best possible subset with respect to the considered objective function. However, in contrast to the discussion for the VRPTW, here we restrict our attention to the single vehicle case. This means that there is no parallel distribution, only production and distribution may happen concurrently.

5.1.2 MIP formulation for the SVMTP

In our mathematical programming formulation for the SVMTP we use the following set of decision variables in order to determine optimal production and distribution sequences:

- Binary variables $Y_{i,j}^r$ for all arcs $(i,j) \in A$ and tours $r \in K$, where $Y_{i,j}^r = 1$ means that tour r contains arc (i,j), and $Y_{i,j}^r = 0$ otherwise.
- Binary variables S_i^r for all customers $i \in C$ and tours $r \in K$, where $S_i^r = 1$ means that customer i is served on tour r, and $S_i^r = 0$ otherwise.
- Non-negative variables τ_i^r for all customers $i \in C$ and tours $r \in K$, representing the delivery time at customer i on tour r. If customer i is not served on tour r, we set $\tau_i^r = 0$.
- Non-negative variables c_i for all customers $i \in C$, representing the production completion time for the order placed by customer i. The value of c_i is meaningless if customer i is not served on any tour.

Using the above decision variables, the mixed integer programming model for the SVMTP can now be stated as follows.

$$\text{Maximize} \quad \sum_{r \in K} \sum_{i \in C} p_i S_i^r - \alpha \sum_{r \in K} \sum_{(i,j) \in A} t_{i,j} Y_{i,j}^r \tag{5.1a}$$

subject to

Set $S_i^r = 1$, if customer i is supplied on tour r:

$$\sum_{(i,j)\in A} Y_{i,j}^r = S_i^r \qquad (i \in C, r \in K) \tag{5.1b}$$

Visit each customer at most once:

$$\sum_{r\in K} S_i^r \leq 1 \qquad (i \in C) \tag{5.1c}$$

Vehicle flow constraints:

$$\sum_{\{h|(h,i)\in A\}} Y_{h,i}^r = \sum_{\{j|(i,j)\in A\}} Y_{i,j}^r \qquad (i \in C, r \in K) \tag{5.1d}$$

$$\sum_{j=1}^{n+1} Y_{0,j}^r = \sum_{i=0}^{n} Y_{i,n+1}^r = 1 \qquad (r \in K) \tag{5.1e}$$

Vehicle capacity constraints:

$$\sum_{i\in C} p_i S_i^r \leq q \qquad (r \in K) \tag{5.1f}$$

Travel time constraints (with M_1 being a sufficiently large constant):

$$\tau_i^r + t_{i,j} - \tau_j^r \leq M_1(1 - Y_{i,j}^r) \qquad ((i,j) \in A, r \in K) \tag{5.1g}$$

Production time constraints (with M_2 being a sufficiently large constant):

$$c_i + p_j - c_j \leq M_2\Big(1 - \sum_{r\in K} Y_{i,j}^r\Big) \qquad ((i,j) \in A) \tag{5.1h}$$

Production between tour starts (with M_3 and M_4 being sufficiently large constants):

$$c_i - \tau_0^r \leq M_3(1 - S_i^r) \qquad (i \in C, r \in K) \tag{5.1i}$$

$$\tau_0^{r-1} - c_i \leq M_4(1 - S_i^r) \qquad (i \in C, r \in K \setminus \{1\}) \tag{5.1j}$$

Time window constraints:

$$a_i S_i^r \;\leq\; \tau_i^r \;\leq\; b_i S_i^r \qquad (i \in C, r \in K) \tag{5.1k}$$

Tour inter-connection constraints:

$$\tau_{n+1}^{r-1} \leq \tau_0^r \qquad (r \in K \setminus \{1\}) \tag{5.1l}$$

$$\tau_0^{r-1} + \sum_{i \in C} p_i S_i^r \leq \tau_0^r \qquad (r \in K \setminus \{1\}) \tag{5.1m}$$

Lifespan constraints:

$$\sum_{r \in K} \tau_i^r - c_i \leq B \qquad (i \in C) \tag{5.1n}$$

Using the variables:

$$Y_{i,j}^r \in \{0,1\} \qquad ((i,j) \in A, r \in K) \tag{5.1o}$$

$$S_i^r \in \{0,1\} \qquad (i \in C, r \in K) \tag{5.1p}$$

$$\tau_i^r \geq 0 \qquad (i \in N, r \in K) \tag{5.1q}$$

$$c_i \geq 0 \qquad (i \in N) \tag{5.1r}$$

The objective function (5.1a) to be maximized consists of a weighted difference between the satisfied demands and the sum of travel times for all tours. The weighting parameter for the travel times is called $\alpha > 0$. If small values are chosen for α, solutions will tend to serve more customers, possibly at the expense of longer travel times. On the other hand, if α is large, solvers will put more effort into the reduction of travel times, possibly restricting the number of served customers.

Another goal to be achieved by an optimal solution may also consist in minimizing the number of vehicle tours needed. For now we will not include this goal in the objective function of our model, but point out that one can solve the model for different values of k, thus restricting the number of generated tours for the vehicle. Note that the search for an appropriate value of k can be performed using a binary search method. Clearly, $k = n$ is an upper bound for the number of tours because this would allow each customer to be served by a separate tour.

In the following we explain the main differences between the single-tour model of Section 4.2 and the multi-tour model presented above. The capacity constraints in inequality (5.1f) state that each vehicle tour can only serve a total demand of up to q units. Due to the added third index r for variables $Y_{i,j}^r$, the production time constraints (5.1h) need to contain the term $\sum_{r \in K} Y_{i,j}^r$, specifying whether arc (i,j) is included in tour r or not. If customer i is served on tour r, i.e. if $S_i^r = 1$, then the right-hand sides of inequalities (5.1i) and (5.1j) both evaluate to zero, which forces the production

completion time of i to be placed between the starts of tour r and $r-1$. Together with the production time constraints (5.1h), this implies that at most one order is produced at any time.

The tour inter-connection constraints (5.1l) and (5.1m) ensure proper sequencing of the tours for the vehicle: tour r can only start when the preceding tour $r-1$ has ended and the duration between the consecutive tour starts must be at least as large as the time required for producing the goods for tour r. The lifespan constraints (5.1n) use the value $\sum_{r \in K} \tau_i^r$, which specifies the delivery time of order i. This is because τ_i^r is forced to be zero by the time window constraints (5.1k), if i is not served on tour r (i.e. $S_i^r = 0$). Thus, at most one of the variables $\tau_i^1, \ldots, \tau_i^k$ can take on a strictly positive value.

5.2 A model for multiple vehicles

In this section we study a further generalization of the single vehicle model with multiple tours, where a fleet of vehicles is responsible for the deliveries. We present an extended mixed integer programming formulation and a heuristic method to tackle this problem.

5.2.1 The extended model

Using several vehicles for the deliveries represents a natural extension of the integrated production and distribution model for the SVMTP that we discussed in the previous section. Instead of assuming a single vehicle, now a **fleet of delivery vehicles** $V := \{1, \ldots, m\}$ is available to serve a set of possible tours $K := \{1, \ldots, k\}$, where some of these tours may be empty (i.e. unneccessary) in a specific solution. This leads to the **multi-vehicle, multiple tours, variable sequences problem (MVMTP)**.

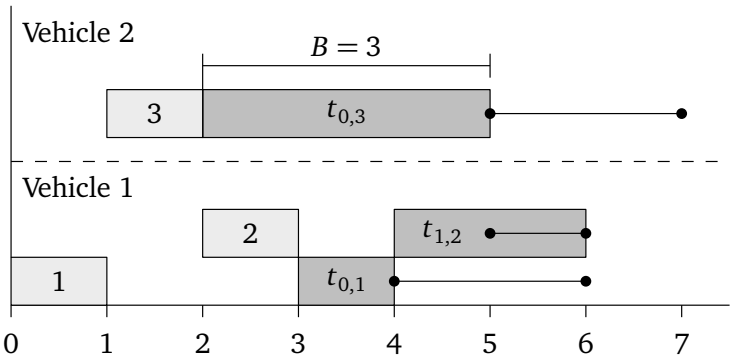
In the MVMTP, all vehicles and customers share the single, sequential production facility. In the next section we argue that it is generally not optimal to produce all orders for each planned vehicle tour consecutively, but instead one has to consider interleaving of production for different tours.

5.2.2 Interleaved production for multiple vehicles

In the single vehicle scenario it is clear that the set of orders belonging to the next distribution tour can be produced consecutively without idle times directly before the

start of this tour. As we have shown in Chapter 4, the distribution sequence does not necessarily need to be equal to the production sequence. Obviously, in order to make the products expire as late as possible, the production should also be started as late as possible such that the customer time windows can still be obeyed.

In the multi vehicle case, however, we can no longer assume that an optimal production/distribution schedule may be generated in such a way that all orders belonging to a specific vehicle tour must be produced without **interleaving** by production phases for other tours (of other vehicles). In the following we will argue that in the presence of limited lifespans, interleaved production is sometimes necessary.



t_{ij}	1	2	3
0	1	2	3
1		2	4
2			3

i	1	2	3
p_i	1	1	1
a_i	4	5	5
b_i	6	6	7

Figure 5.2: Interleaving of production for 2 vehicles. It is impossible to produce the orders for vehicle 1 consecutively, if all customers have to be supplied.

Example 3 Consider the small example shown in Figure 5.2. It consists of $n = 3$ customers, $m = 2$ vehicles and assumes a lifespan of $B = 3$ time units. Moreover, travel times t_{ij}, demands p_i, and time windows $[a_i, b_i]$ are given. Note that the travel times obey the triangle inequality. Regarding the total satisfied demand objective function, the figure depicts an optimal production and distribution schedule for this example because all customers are supplied and the schedule is feasible. Due to the incompatible time windows and travel times, it is impossible to serve customers 1 and 3 with the same vehicle on a single tour. The same holds true for the combination of orders for customers 2 and 3. However, serving customers 1 and 2 by a single vehicle on a single tour is admissible as shown in the figure. Using a vehicle more than once is not reasonable in this example. This means that customer 3 must be served by a dedicated transporter (if all orders are produced). It is easy to see that we cannot produce orders 1 and 2 consecutively (for vehicle 1) and order 3 (for vehicle 2) afterwards since the production for order 3 is forced to start at time 1. The reason for this is that order 2 must not be completed earlier than time 3 (due to its lifespan)

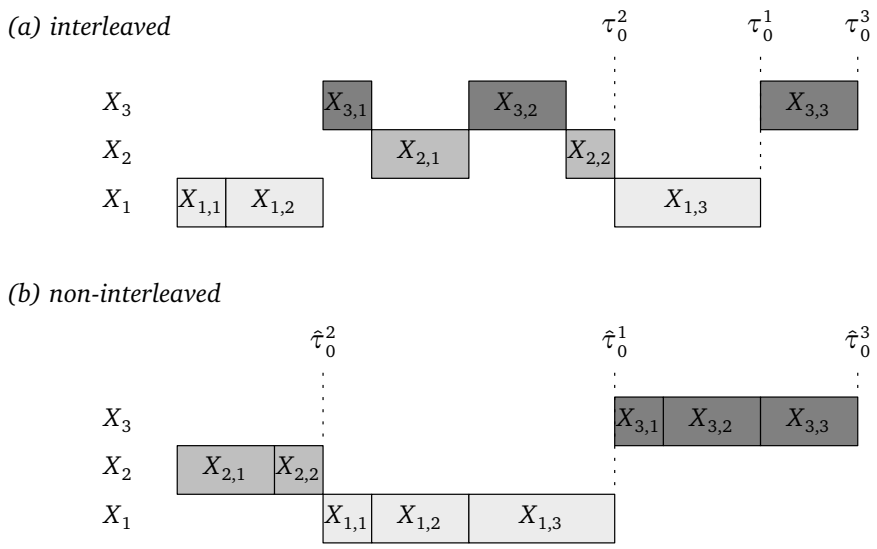


Figure 5.3: Example for $\mathcal{X} = \{X_1, X_2, X_3\}$: Reordering the interleaved production schedule (a) to obtain non-interleaved schedule (b) is always feasible if $B = \infty$.

and 3 must not be completed earlier than time 2. As a consequence, in an optimal schedule the production for vehicle 1 must be interleaved with that for vehicle 2. ◊

The theorem given below captures the fact that interleaving the production of orders for different tours cannot admit better production and distribution plans, if the product lifespan is umlimited. For a formal description, let us consider an optimal production and distribution plan $\sigma = (\sigma^P, \sigma^D)$ for an instance of the MVMTP. Moreover, denote by $\mathcal{T}_\mu := \{\mathcal{T}_{\mu,1}, \ldots, \mathcal{T}_{\mu,l_\mu}\}$ the set of tours scheduled for vehicle $\mu = 1, \ldots, m$ and let $\mathcal{T} := \bigcup_\mu \mathcal{T}_\mu$ be the set of all tours.

Theorem 8 *Let $\mathcal{X} \subseteq \mathcal{T}$ be an arbitrary subset of tours such that the production sequence σ^P for the contained orders is interleaved. Under the assumption $B = \infty$, it is always possible to reorder the jobs in σ^P such that no interleaving of orders belonging to different tours occurs, i.e. the production for each tour in $\mathcal{T}$ is carried out consecutively without idle times and the resulting schedule remains feasible.*

Proof Let $\mathcal{X} := \{X_1, \ldots, X_r\}$ be a subset of tours with the above-mentioned properties. We assume w.l.o.g. that these tours are numbered according to ascending transportation start times τ_0^i in σ. Then, the set of all jobs $j \in \bigcup_{i=1}^r X_i$ can be reordered in σ^P in such a way that all jobs belonging to a tour X_i are produced consecutively without idle times before all jobs belonging to a succeeding tour X_j, with $j > i$ (see Figure 5.3). In

doing so, due to the sorting, each tour may start earlier or at the same time as before, i.e. $\hat{\tau}_0^i \leq \tau_0^i$. Since the product lifespan is unlimited and none of the transportation start times is delayed, the resulting production and distribution plan is again feasible. By repeating this argument for each such subset $\mathcal{X}$ we obtain a feasible plan without interleaved jobs in the production sequence. □

Together with Example 3, this theorem shows that starting individual tours earlier may violate lifespan constraints, but it does not have any influence on time window feasibility. Thus, we can only obtain better schedules by considering interleaved production plans, if the lifespan is limited.

Lemma 9 *Allowing an interleaved production sequence may only improve solution quality if the product lifespan is limited.*

5.2.3 MIP formulation for the MVMTP

In this section we propose a MIP formulation for the MVMTP, which represents the most general integrated production and distribution model that we study in this thesis. It is composed of the conditions (5.2) which are similar to the previously discussed restrictions for the SVMTP in model (5.1). We will focus on the main differences and extensions that are required to deal with several capacitated vehicles in the transportation planning. Some of these extensions, especially the restrictions for handling the sequencing of vehicle tours, are inspired by a model for the VRPTW proposed by Azi et al. (2010).

The model (5.2) distinguishes between the production sequence and a set of tours $K := \{1, \ldots, k\}$, which are carried out by a set $V := \{1, \ldots, m\}$ of vehicles. Just like in the model for the SVMTP, some of the tours may not be needed (i.e. they remain empty). Moreover, depending on the orders, not every available vehicle may be required to perform the deliveries.

The production sequence is represented by binary variables $X_{i,j}$ for each $(i, j) \in A$, where $X_{i,j} = 1$ means that order i is the immediate predecessor of order j, although there may be an idle time between them. Constraint (5.2b) establishes that the order for each supplied customer $i \in C$ is produced. Furthermore, since the $X_{i,j}$ are supposed to represent a sequence, the model contains further production flow constraints in (5.2e) and (5.2f). As the production machine can handle at most one job at any time, production times for orders may not overlap. This is ensured by restrictions (5.2k), where the difference between two consecutive production completion times c_i and c_j is set to be at least the production time of j.

Here is a comprehensive overview of the decision variables used in the model:

- Binary variables $X_{i,j}$ for all arcs $(i,j) \in A$, where $X_{i,j} = 1$ means that i is produced directly before j, and $X_{i,j} = 0$ otherwise.
- Binary variables $Y^r_{i,j}$ for all arcs $(i,j) \in A$ and tours $r \in K$, where $Y^r_{i,j} = 1$ means that tour r contains arc (i,j), and $Y^r_{i,j} = 0$ otherwise.
- Binary variables $Z_{r,s}$ for all tours $r,s \in K$ with $r < s$, where $Z_{r,s} = 1$ means that tour r directly precedes tour s for one of the vehicles, and $Z_{r,s} = 0$ otherwise.
- Binary variables D^r_v for all tours $r \in K$ and all vehicles $v \in V$, where $D^r_v = 1$ means that tour r is served by vehicle v, and $D^r_v = 0$ otherwise.
- Binary variables S^r_i for all customers $i \in C$ and tours $r \in K$, where $S^r_i = 1$ means that customer i is served on tour r, and $S^r_i = 0$ otherwise.
- Non-negative variables τ^r_i for all customers $i \in C$ and tours $r \in K$, representing the delivery time at customer i on tour r. If customer i is not served on tour r, we set $\tau^r_i = 0$.
- Non-negative variables c_i for all customers $i \in C$, representing the production completion time for the order placed by customer i. The value of c_i is meaningless if customer i is not served on any tour.

$$\text{Maximize} \quad \sum_{r\in K}\sum_{i\in C} p_i S^r_i - \alpha \sum_{r\in K}\sum_{(i,j)\in A} t_{i,j} Y^r_{i,j} \tag{5.2a}$$

subject to

Set $S^r_i = 1$, if customer i is supplied on tour r:

$$\sum_{\{j|(i,j)\in A\}} Y^r_{i,j} = S^r_i \qquad (i \in C, r \in K) \tag{5.2b}$$

If customer i is supplied, the corresponding order must be produced:

$$\sum_{\{j|(i,j)\in A\}} X_{i,j} = \sum_{r\in K} S^r_i \qquad (i \in C) \tag{5.2c}$$

Each customer is supplied at most once:

$$\sum_{r \in K} S_i^r \leq 1 \qquad (i \in C) \qquad (5.2d)$$

Production flow constraints:

$$\sum_{\{h|(h,i) \in A\}} X_{h,i} = \sum_{\{j|(i,j) \in A\}} X_{i,j} \qquad (i \in C) \qquad (5.2e)$$

$$\sum_{j=1}^{n+1} X_{0,j} = \sum_{i=0}^{n} X_{i,n+1} = 1 \qquad (5.2f)$$

Vehicle flow constraints:

$$\sum_{\{h|(h,i) \in A\}} Y_{h,i}^r = \sum_{\{j|(i,j) \in A\}} Y_{i,j}^r \qquad (i \in C, r \in K) \qquad (5.2g)$$

$$\sum_{j=1}^{n+1} Y_{0,j}^r = \sum_{i=0}^{n} Y_{i,n+1}^r = 1 \qquad (r \in K) \qquad (5.2h)$$

Vehicle capacity constraints:

$$\sum_{i \in C} p_i S_i^r \leq \sum_{v \in V} q_v D_v^r \qquad (r \in K) \qquad (5.2i)$$

Travel time constraints (with M_1 being a sufficiently large constant):

$$\tau_i^r + t_{i,j} - \tau_j^r \leq M_1(1 - Y_{i,j}^r) \qquad ((i,j) \in A, r \in K) \qquad (5.2j)$$

Production time constraints (with M_2 being a sufficiently large constant):

$$c_i + p_j - c_j \leq M_2(1 - X_{i,j}) \qquad ((i,j) \in C \times C) \qquad (5.2k)$$

Time window constraints:

$$a_i S_i^r \;\leq\; \tau_i^r \;\leq\; b_i S_i^r \qquad (i \in C, r \in K) \qquad (5.2l)$$

Lifespan constraints:

$$\sum_{r \in K} \tau_i^r - c_i \leq B \qquad (i \in C) \qquad (5.2m)$$

Tour sequencing constraints (with M_3 being a sufficiently large constant):

$$\tau^r_{n+1} - \tau^s_0 \leq M_3(1 - Z_{r,s}) \qquad (r,s \in K, r < s) \qquad (5.2n)$$

$$\sum_{r \in K} \sum_{\{s \in K | r < s\}} Z_{r,s} \geq |K| - |V| \qquad (5.2o)$$

Each tour has at most one successor and one predecessor:

$$\sum_{s \in K, r < s} Z_{r,s} \leq 1 \qquad (r \in K) \qquad (5.2p)$$

$$\sum_{r \in K, r < s} Z_{r,s} \leq 1 \qquad (s \in K) \qquad (5.2q)$$

If r and s are served by the same vehicle, both tours must not overlap in time:

$$\tau^r_{n+1} - \tau^s_0 \leq M_4(2 - D^r_v - D^s_v) \qquad (r,s \in K, r < s) \qquad (5.2r)$$

Produce orders before tour starts (with M_5 being a suff. large constant):

$$c_i - \tau^r_0 \leq M_5(1 - S^r_i) \qquad (i \in C, r \in K) \qquad (5.2s)$$

Each tour is served by exactly one vehicle:

$$\sum_{v \in V} D^r_v = 1 \qquad (r \in K) \qquad (5.2t)$$

If $Z_{r,s} = 1$ then r and s must be served by the same vehicle (M_6 suff. large const.):

$$M_6(Z_{r,s} - 1) \leq \sum_{v \in V} v \cdot (D^r_v - D^s_v) \leq M_6(1 - Z_{r,s}) \qquad (r,s \in K, r < s) \qquad (5.2u)$$

Using the variables:

$$\begin{array}{rrrr}
X_{i,j} \in \{0,1\} & ((i,j) \in A) & S^r_i \in \{0,1\} & (i \in C, r \in K) \\
Y^r_{i,j} \in \{0,1\} & ((i,j) \in A, r \in K) & \tau^r_i \geq 0 & (i \in N, r \in K) \\
Z_{r,s} \in \{0,1\} & (r,s \in K, r < s) & c_i \geq 0 & (i \in C) \\
D^r_v \in \{0,1\} & (v \in V, r \in K) & &
\end{array}$$

In contrast to the model for the SVMTP, now the planned tours are generally not required to be consecutive because deliveries are performed concurrently by the vehicles. Of course, if a set of tours are served by the same vehicle, these tours must not overlap in time. Observe that if one vehicle serves k tours, there will be $k-1$ intermediate depot returns. More generally, if exactly $|V| = m$ vehicles perform $|K| = k$ tours with $m \leq k \leq n$, there are $k-m$ transitions between tours, i.e. intermediate depot returns for some vehicles. Consequently, if we are allowed to employ *at most* m vehicles, we will need to schedule *at least* $k-m$ tour transitions. Our model reflects this observation by introducing binary variables $Z_{r,s}$ for each pair of tours $r,s \in K$ with $r < s$, which indicate whether or not tour s immediately succeeds tour r in the schedule of one of the vehicles. Using these variables, proper tour sequencing is ensured by restrictions (5.2n) and (5.2o). The latter constraint specifies that the number of $Z_{r,s}$ variables set to one is at least the minimum number of tour transitions. There may also be more transitions (intermediate depot returns) unless the entire set of vehicles is actually used in a solution. If $Z_{r,s} = 1$, then inequality (5.2n) implies that the start of tour s must happen after the end of tour r. Furthermore, we ensure that each tour is assigned at most one successor (resp. predecessor) tour via constraints (5.2p) and (5.2q).

The computed tours must be assigned to vehicles that are ready to perform them. We integrate this assignment in our model by adding binary variables D_v^r for each vehicle $v \in V$ and each tour $r \in K$, where $D_v^r = 1$ if and only if tour r is served by vehicle v. Since vehicles may have different capacities (heterogeneous vehicle fleet), the vehicle capacity constraints must be extended as shown in (5.2i), where parameter q_v denotes the capacity of vehicle v. The previously discussed requirement that tours served by the same vehicle must not overlap in time, is reflected by inequalities (5.2r). Furthermore, the assignment of values to variables D_v^r must be consistent, i.e. each tour must be served by exactly one vehicle (5.2t). Finally, it is necessary to state by constraints (5.2u) that an intermediate depot return between tours $r,s \in K$, $r < s$, can be scheduled by setting $Z_{r,s} = 1$, only if r and s are served by the same vehicle. This is achieved – together with the "consistency" constraints – by requiring that $\sum_{v \in V} v \cdot (D_v^r - D_v^s) = 0$ if $Z_{r,s} = 1$, because in this case the summation can only evaluate to zero if $D_v^r = 1$ and $D_v^s = 1$ for the same vehicle v.

5.3 An ILS-based heuristic for the MVMTP

In this section we study a heuristic solution approach to solve the MVMTP. Since the iterated local search method that we implemented for the SVDSP (see Section 4.3.3, p. 67) is capable of computing high-quality solutions (see Section 6.5.2), we have

also addressed the more general MVMTP using this paradigm. Before dealing with the actual implementation of the algorithm we discuss the corresponding solution representation and the method we use for production planning.

5.3.1 Solution representation

In order to apply local search operations to solutions for the MVMTP we use the following solution representation. A **multi-vehicle production and distribution schedule** is a pair $\sigma = (\sigma^P, \sigma^D)$ with the following properties.

- The production schedule σ^P is represented by a mapping $s : C \to \mathbb{N}_0$ of customers to start times. For unsupplied customers $i \in C$ we set $s_i := s(i) := 0$. As preemption is not allowed, the production completion times are $c_i := s_i + p_i$.
- The distribution schedule σ^D is represented by a set of **routes** $r = 1, \ldots, m$, where $m = |V|$ is the number of available vehicles. Thus, each vehicle is assigned to a corresponding route. Each route r is a sequence of customers with dummy nodes for the start- and end-depot at the beginning and at the end. Furthermore, intermediate depot returns are allowed, just like in the model for the SVMTP (see Section 5.1 on p. 72). These depot returns partition each route into several **tours** for the assigned vehicle. The amount of goods delivered on each tour of any route r is limited by the vehicle capacity q_r.

A graphical overview of the solution representation is given in Figure 5.4.

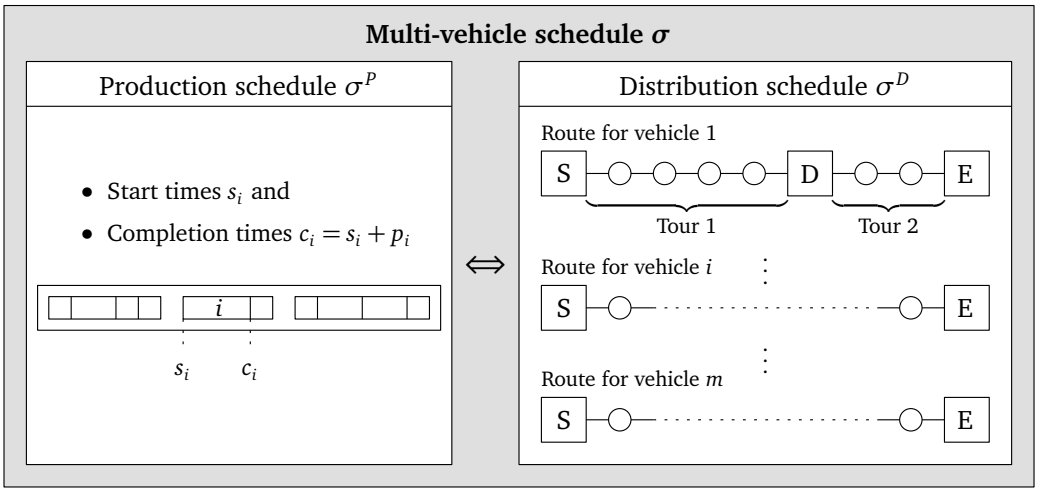


Figure 5.4: Solution representation for MVMTP heuristic

5.3.2 Production planning

The insertion operations that we use in our implementation of ILS primarily consider the current distribution tours because for most instances the customer time windows represent the most restrictive constraints for finding feasible schedules. The central idea consists in deriving a feasible production schedule from a given set of feasible routes for the vehicles.

Suppose a feasible distribution schedule σ^D is given together with corresponding delivery times τ_i for all supplied customers $i \in C$. Due to the lifespan constraints the production of order i must not begin earlier than at time $\tau_i - B - p_i$. Thus we can derive earliest start times (**release dates**) $r_i := \max\{0,\ \tau_i - B - p_i\}$ for the production of each order i. Furthermore, we can also compute latest production completion times (**due dates**) d_i for each order i based on the observation that each order must be completed until the associated delivery tour starts. This way we obtain time windows $[r_i, d_i]$ for any production schedule σ^P that is feasible with respect to the given set of delivery tours and times.

In the *One-machine Sequencing Problem* (OMS) a single machine is given that processes a number of jobs $i = 1, \ldots, n$ for a duration of p_i time units without preemption. Each job i is associated with a corresponding time window $[r_i, d_i]$ within which it must be processed. The OMS has been proved to be $\mathcal{NP}$-hard in the strong sense by Lenstra et al. (1977). If we denote by s_i the processing start times, the OMS consists in finding such values s_i subject to the following restrictions:

$$\begin{aligned} s_i &\geq r_i && (i = 1, \ldots, n) \\ s_i + p_i &\leq d_i && (i = 1, \ldots, n) \\ s_j + p_j \leq s_i \ \vee\ s_i + p_i &\leq s_j && (i, j = 1, \ldots, n,\ i \neq j) \end{aligned}$$

The former constraints ensure that each job is scheduled within its time window, whereas the latter disjunctive constraints enforce that at most one job is processed at any time. We can state a corresponding optimization problem OMS_{opt} by using artificial delivery durations (tails) $q_i = T - d_i$, where $T := \max_i\{d_i\}$ can be interpreted as the planning horizon, i.e. the latest time at which all processing must be finished.

$$\begin{aligned} (\text{OMS}_{\text{opt}})\ \text{Minimize} \quad & C_{\max} \\ \text{s.t.} \quad & s_i \geq r_i && (i = 1, \ldots, n) \\ & C_{\max} \geq s_i + p_i + q_i && (i = 1, \ldots, n) \\ & s_j + p_j \leq s_i \ \vee\ s_i + p_i \leq s_j && (i, j = 1, \ldots, n,\ i \neq j) \end{aligned}$$

The objective in OMS_{opt} is to minimize the makespan $C_{\max}$ of the schedule. It is easy to see that the original OMS has a feasible solution if and only if OMS_{opt} has an optimal solution value $C^*_{\max} \leq T$. If this is the case, the values s^*_i determine feasible start times. In the following we describe Carlier's algorithm for solving the OMS. We use this method for production planning in our implementation, based on an idea used by Wendt (2009) in a two-stage routing-first, production-second heuristic.

5.3.3 Carlier's algorithm for one-machine sequencing (OMS)

In order to solve the OMS, Carlier (1982) developed a branch and bound algorithm which uses a heuristic schedule generation method initially proposed by Schrage (1970, 1971). The basic idea of the heuristic is to begin at the earliest release date $t = \min_j\{r_j\}$ and to iteratively schedule a ready job i having the largest tail value q_i. Schrage's algorithm has a worst-case performance ratio of 2, which could subsequently be improved to 1.5 by a modification due to Potts (1980).

Carlier uses the resulting Schrage schedule to derive lower bounds and a branching scheme. Algorithm 5.1 presents pseudocode for the heuristic, which may be implemented to run in $\mathcal{O}(n \log n)$ time, as Carlier (1982) demonstrates.

Algorithm 5.1 Schrage algorithm

```
procedure SCHRAGE(n, p_i, r_i, q_i)
    S ← ∅ ;  S̄ ← {1, ..., n}
    t ← min_{j∈S̄}{r_j}
    while S̄ ≠ ∅ do
        I ← { j ∈ S̄ | r_j ≤ t }
        i ← argmax_{j∈I}{q_j}
        S ← S ∪ {i}
        S̄ ← S̄ \ {i}
        s_i ← t
        t ← max { t + p_i, min_{j∈S̄}{r_j} }
```

The schedule given by the start times s_i which are computed by the Schrage algorithm contains at least one critical path $P = (i_1, \ldots, i_k)$ with a critical set $J \subseteq \{1, \ldots, n\}$ and $r_{i_1} + \sum_{i \in J} p_i + q_{i_k} = C_{\max}$, such that P is maximal in the sense that it cannot be extended by the immediate predecessor and/or successor in the Schrage schedule while retaining the critical path property. Furthermore, there may be a critical job $c \in P$ with $q_c < q_{i_k}$. The correctness of Carlier's branch and bound algorithm is based on Theorems 10 and 11.

Theorem 10 *Let $C_{\max}$ be the makespan of a given Schrage schedule s. If s is not optimal, there is a critical job c and a critical set J with*

$$\min_{i \in J}\{r_i\} + \sum_{i \in J} p_i + \min_{i \in J}\{q_i\} > C_{\max} - p_c.$$

The distance from $C_{\max}$ to the optimum is less than p_c. Moreover, in an optimal schedule, job c is either processed before or after all jobs in J.

Theorem 11 *Let s be a Schrage schedule. If s contains a critical path $P = (i_1, \ldots, i_k)$ with $q_j \geq q_{i_k}$ for all $j \in P$, then s is optimal.*

Initially, the branch and bound algorithm computes the Schrage-schedule for a given problem instance. If this schedule contains a critical job c and a critical set J according to Theorem 10, the problem is split into two subproblems, in which c is scheduled either before or after all jobs in J, respectively. After that, lower bounds are computed for the subproblems (by considering relaxations based on allowing preemption) and if these bounds do not imply a cut-off, the subproblems are added into a priority-queue that is organized according to the lower bound values. The items in the queue are analyzed recursively, using a variant of the Schrage-algorithm that respects the additional sequencing constraints imposed by the placement of the critical jobs and sets.

For the purpose of production planning we do not necessarily need to find a minimum-makespan solution to the OMS because it is sufficient to ensure the existence of a feasible production schedule with a makespan of $C_{\max} \leq T = \max_i\{d_i\}$. Consequently, we may interrupt Carlier's algorithm as soon as such a solution is found and use the obtained schedule for the production phase. Our computational experiments showed that using this adaptation is crucial to reduce required runtimes.

5.3.4 Applying ILS to the MVMTP

In the following we describe our implementation of iterated local search for the MVMTP, based on the previously discussed solution representation. Remember that the latter includes starting times for the production phase and sequences of customers and depot visits for the distribution phase.

Initially we assume that each vehicle $v \in V$ is assigned an empty route $\rho_v = (0, n+1)$, i.e. only the dummy nodes for start and end are included. The heuristic now mainly concentrates on the distribution phase by applying the following concept: Choose

a currently unsupplied customer and compute its cost of insertion at each possible position in each route ρ. Also check whether a given insertion is feasible by deriving production time windows and a production schedule via Carlier's algorithm. Finally execute a least-cost, feasible insertion for the considered customer. Repeating this strategy leads to a local optimum with respect to the given insertion-neighborhood in the sense that either all customers are supplied or none of the still unsupplied customers can be added without making at least one of the delivery routes infeasible. In order to leave this area of the search space, a diversification procedure is required that is called "shaking" in ILS (c.f. Section 4.3.3). Our implementation alternately applies the insertion and shaking procedures to explore the set of feasible solutions. We will now explain these procedures in more detail.

At first, we give an overview of the operations that are applied to a given schedule σ in the insertion and shaking procedures:

- INSERT(c, ρ, j): Insert customer $c \in C$ into route $\rho \in \{1, \ldots, m\}$ at position $j \in \{1, \ldots, |\rho| - 1)\}$, where $|\rho|$ denotes the number of visits (customer and depot visits) in route ρ. Note that for an empty route $\rho = (0, n+1)$ we have $|\rho| = 2$.
- REMOVE(c, ρ): Remove customer $c \in C$ from its containing route $\rho \in \{1, \ldots, m\}$.
- INSERTDEPOT(ρ, j) Insert intermediate depot visit into route $\rho \in \{1, \ldots, m\}$ at position j. This operation should not cause successive (unnecessary) depot visits, i.e. a position j is forbidden if there is a depot visit at $j-1$ or j prior to the insertion.
- REMOVESUCCESSIVEDEPOTS(ρ): If route $\rho \in \{1, \ldots, m\}$ contains successive depot visits i and j, where at least one of i and j is an intermediate depot visit, remove all such unnecessary (intermediate) visits.

Insertion

Insertion represents the improvement phase of the iterated local search, where unsupplied customers are added one by one to the current solution as long as the latter remains feasible. Algorithm 5.2 provides a pseudocode summary for this procedure, which constitutes the central part of our implementation.

The first step in procedure INSERTIONLS(σ) is to make a list of currently unselected customers U. This list determines the order in which customers are tested for insertion. For our computational experiments we used the following strategies:

Algorithm 5.2 ILS for the MVMTP: Insertion

```
1:  procedure InsertionLS(σ)
2:      U ← MakeListOfUnselectedCustomers(σ)
3:      for each customer c ∈ U do
4:          I_c ← ∅                                          ▷ Feasible insertions for c
5:          for each route ρ = 1, ..., m in σ do
6:              CheckDepotInsertion(ρ)
7:              for each position j in ρ do
8:                  Insert c at position j in route ρ
9:                  LDT_ρ ← ComputeLatestDeliveryTimes(ρ)
10:                 (r, d) ← ComputeProductionTimeWindows(LDT_1, ..., LDT_m)
11:                 σ^P ← Carlier(σ, r, d)
12:                 if σ' ← (σ^P, LDT) is a feasible schedule then
13:                     I_c ← I_c ∪ {σ'}
14:                 Remove c from route ρ
15:         if I ≠ ∅ then
16:             Choose i ∈ I_c with Value(i) ≥ Value(i') for all i' ∈ I_c
17:             σ ← i
```

(a) Sort the set of unselected customers according to non-decreasing ratios $\frac{b_i - a_i}{p_i^2}$, $i \in C$, so that customers with narrow time windows and/or large demands are initially chosen as insertion candidates. The reasoning behind this is that more flexible customers with wide time windows and/or rather small demands are examined later on and are then hopefully easier to insert. Note that the ratios are well-defined because $p_i > 0$ for all $i \in C$.

(b) Sort all unselected customers $i \in C$ according to non-decreasing delivery deadlines b_i as an adaptation of the heuristic earliest-deadline-first rule.

(c) Use a random permutation of all unselected customers with the aim of introducing even more diversification than shaking alone can provide.

The procedure goes on by selecting the next unsupplied customer c from list U, and then tries to tentatively insert c at each possible position of each route ρ in turn. In step 9, latest delivery times LDT_ρ for the altered route ρ are computed. This is done in a similar manner as discussed in the single-vehicle model in Section 3.3, p. 33. In the multi-tour case, however, delaying potentials (see Def. 2, p. 35) must be computed for *each tour* of a route ρ because the vehicle is allowed to wait at the depot for reloading. Waiting at the depot may be beneficial since this creates more flexibility for production planning and helps to construct delivery schedules so that the goods expire as late as possible. For this reason, the times in LDT_ρ are computed for each tour by the usual

ESS method, where the individual tour starts are postponed by the corresponding delaying potentials.

Figure 5.5: Earliest and latest delivery times for a route consisting of 2 tours

An example of a resulting schedule is illustrated in Figure 5.5, which shows earliest and latest delivery times for a route including four customers and one intermediate depot return (Depot 1). As mentioned above, such multi-tour schedules may contain waiting times at the depot (indicated by the black box in the figure). In the example, the earliest tour starts are given by times 0 and 8, respectively, whereas the latest tour starts are 7 and 15. Note that, especially for the latest delivery times, the visits are planned via ESS for every single tour of the route, each starting at the latest possible time. The waiting time at the depot (time 13 to 15) is only possible in this example because the time windows allow for this.

As soon as the latest delivery times LDT_ρ have been computed for each non-empty route ρ, the insertion procedure derives production time windows $[r_i, d_i]$ for each supplied customer $i \in C$ in step 10 as described in Section 5.3.2. After that, the Carlier algorithm is called to solve the resulting OMS for the production schedule (step 11). If a feasible solution to the OMS is found, we store the corresponding multi-route schedule in a set I_c of feasible insertions for customer c and remove c from ρ again to restore the initial state for the next iteration. After collecting all feasible insertions for c, we choose such an insertion $i \in I_c$ with the largest value (step 16) and update the current solution σ.

In order to compute the **value of a multi-vehicle schedule σ** we propose the following method. Let $C_\sigma \subseteq C$ denote the subset of customers who are supplied in σ and let $V_\sigma \subseteq V$ be the subset of vehicles which are assigned non-empty routes in σ.

Furthermore, let t_j, $j = 1, \ldots, m$, denote the total travel time needed for route j. Then the value of σ is defined as

$$\textsc{Value}(\sigma) := \sum_{i \in C_\sigma} p_i - \alpha_1 \sum_{j=1}^{m} t_j - \alpha_2 \sum_{v \in V_\sigma} q_v \,, \qquad \alpha_1, \alpha_2 > 0. \tag{5.3}$$

Multiple goals are addressed by this objective function. Besides maximizing the total satisfied demand, the aim is to minimize the weighted total distance traveled by the vehicles and the weighted sum of vehicle capacities employed for the deliveries. If the vehicles are homogeneous, i.e. all have the same capacity, this results in minimizing the number of vehicles used.

As the vehicles are allowed to perform multiple tours per route, InsertionLS must occasionally consider inserting intermediate depot returns. This is performed in procedure CheckDepotInsertion in step 6. Inserting a depot visit always splits a specific tour θ of a route ρ into two non-empty tours θ_1, θ_2. Firstly, the method checks whether the current load of each tour $\theta \in \rho$ has come close to the associated vehicle capacity limit. Experiments showed that splitting a tour is worth considering if the load exceeds 85% of the vehicle capacity.

Secondly, a tour θ should be considered a candidate for splitting if the schedule has become so compact that customers along θ receive their orders quite close to the expiry of the goods, i.e. if the **lifespan flexibility** of the tour, defined as

$$\varphi(\theta) := \min_{i \in \theta} \{c_i + B - \tau_i\}$$

falls below a certain threshold. This threshold should depend on the lifespan B; in our implementation we decide to split a tour θ if $\varphi(\theta) < B/10$. Since a further depot return incurs additional traveling time, the route must be checked for feasibility. We determine the actual position at which the depot return is finally inserted by checking all possible positions in the considered tour and by choosing a feasible insertion that incurs the least delay in the delivery time of the directly following customer.

Shaking

Following the ILS strategy, we attempt to escape from local optima by applying a diversification method that removes a varying number of successive customer visits in each route. This procedure is called Shake and it is detailed in Algorithm 5.3.

For each route we compute the maximum number of customers to be removed as the minimum of the route size and the parameter `nrToRemove` (step 3). After that, the

Algorithm 5.3 ILS for the MVMTP: Shaking

1: **procedure** SHAKE(σ, `nrToRemove`, `startPos`)
2: **for** each non-empty route ρ in σ **do**
3: $x \leftarrow \min\{|\rho|$, `nrToRemove`$\}$
4: Beginning at `startPos`, remove x successive customers from ρ (skip depot visits and wrap around, if necessary)
5: Replace successive depot visits in ρ by a single depot visit

removal operation is executed while skipping depot visits and beginning at the start of the route if its end is reached and there are still customers left to be deleted (wrap around, step 4). Since procedure SHAKE removes only (real) customer visits directly, it must also take care of removing possibly caused unnecessary depot visits. The latter are condensed to single depot visits in step 5, if necessary.

The main ILS procedure

The main ILS procedure presented in Algorithm 5.4 constitutes the framework of the heuristic in which the already discussed insertion and shaking procedures are embedded. Furthermore, the framework controls the steering parameters `nrToRemove` and `startPos` that are used in the diversification phase of the search.

Algorithm 5.4 ILS for the MVMTP: Main procedure

procedure MULTIVEHICLEILS
  $\sigma^* \leftarrow \sigma \leftarrow$ INSERTIONLS($\emptyset$)
  `nrToRemove` $\leftarrow$ `startPos` $\leftarrow 1$
  repeat
    SHAKE(σ, `nrToRemove`, `startPos`)
    `startPos` $\leftarrow$ `startPos` + `nrToRemove`
    `nrToRemove` $\leftarrow$ `nrToRemove` + 1
    if `startPos` > Size of smallest route **then**
      `startPos` $\leftarrow$ `startPos` − Size of smallest route
    if `nrToRemove` > `Threshold` **then**
      `nrToRemove` $\leftarrow 1$
    INSERTIONLS(σ)
    if VALUE(σ) > VALUE(σ^*) **then**
      $\sigma^* \leftarrow \sigma$
      `nrToRemove` $\leftarrow 1$
  until a stopping criterion is satisfied
  return σ^*

Procedure MultiVehicleILS starts off from an empty schedule $\sigma = \emptyset$ and initializes both of the above named parameters for the shaking phase to 1. Each execution of the following repeat-loop represents an iteration of the heuristic, and the stopping criterion may consist in a preset maximum number of iterations, a maximum runtime in seconds on the hardware used, or in a maximum number of iterations without an improvement of the best known solution. In our computational experiments we used the maximum runtime criterion.

During each iteration, the Shake procedure is applied to the current solution. After that we increase the starting position `startPos` for the removal of customers in each route according to the current value of `nrToRemove`. Moreover, `nrToRemove` is incremented by one. The if-statement in step 8 checks whether `startPos` exceeds the size of the smallest route in the current solution and subtracts the latter, if necessary. Similarly, if `nrToRemove` exceeds a `Threshold` value, it is reset to one. The `Threshold` can be defined in terms of the total number of customers n and/or vehicles m. For example, Vansteenwegen et al. (2009b) use a value of $\frac{n}{3m}$ in a team orienteering setting (TOPTW). The authors report that altering the threshold value between $\frac{n}{m}$ and $\frac{n}{5m}$ does not have a significant effect on the obtained results.

Algorithm 5.4 continues by applying InsertionLS to the incumbent schedule σ. If its value is better than the value of the best known solution σ^*, the latter is updated accordingly. This update also leads to a reset of parameter `nrToRemove` to one, which restricts the following removal process in order to stay in the vicinity of the current (best known) solution.

5.4 Routing First, Production Second

In his diploma thesis, Wendt (2009) analyzes a two-stage heuristic approach called *Routing First, Production Second (RFPS)* to solve the MVMTP.

The distribution scheduling in RFPS is performed by a problem-specific adaptation of a tabu search algorithm proposed by Cordeau et al. (2001) for the VRPTW. Admissible neighborhood moves include insertions (removals) of customers into (from) vehicle routes, as well as opening new tours by adding (removing) intermediate depot visits at the beginning or at the end of existing routes. In contrast to our implementation of ILS, RFPS also allows infeasible schedules as intermediate solutions. Consequently, a penalty function is introduced into the objective function to assess the level of infeasibility of a given schedule. If the schedule if feasible, the penalty value is zero. RFPS manages a tabu list of solution attributes that store information about which customer is served by which vehicle.

In a very similar way compared to ILS, RFPS derives production time windows from a given distribution schedule and solves the resulting one-machine sequencing problem for the production by employing the branch and bound algorithm of Carlier (1982) (c.f. Section 5.3.3).

Since RFPS strives to minimize the costs of a schedule instead of maximizing its value like in ILS, an appropriate cost function must be defined. Using the same notation as in equation (5.3), the costs of a feasible multi-vehicle schedule σ are defined as the overall demand minus the value of σ:

$$\begin{aligned}\textsc{Cost}(\sigma) &:= \sum_{i \in C} p_i - \textsc{Value}(\sigma) \\ &= \sum_{i \in C \setminus C_\sigma} p_i + \alpha_1 \sum_{j=1}^{m} t_j + \alpha_2 \sum_{v \in V_\sigma} q_v \,, \qquad \alpha_1, \alpha_2 > 0. \end{aligned} \tag{5.4}$$

This cost function is always non-negative and can be interpreted as a penalty function for unsupplied demands, travel times, and employed vehicles in σ. Since the costs of σ are defined by negating the value of σ and adding the total customer demand, which is constant for each instance, we can easily transform costs into values and vice versa. This is useful for our comparison of ILS vs. RFPS in Section 6.6 because an optimal solution having maximum value in ILS corresponds to an optimal solution having minimum costs in RFPS.

Chapter 6

Computational Results

In this chapter we report on the results of our computational experiments and describe the observed performance and running times of the algorithms implemented during this research project. Since suitable problem instances for integrated production and distribution problems are not readily available from the literature, the majority of our tests were run on a testbed of self-generated, randomly created instances – based on a set of key parameters.

In Section 6.1 we provide a detailed description of the instance generation process and the controlling parameters involved. The remaining parts of this chapter deal with the specific test cases we considered. Section 6.2 undertakes to quantify the influence of the corrected calculation of maximum transport delays discussed in Section 3.1, whereas Section 6.3 contains an evaluation of the effect of allowing delayed production starts (introduced in Section 3.3) on the performance of branch and bound algorithms for the basic production and distribution model. Results of heuristics for the basic model are discussed in Section 6.4, whereas the proposed heuristics for the models with variable customers sequences (SVESP/SVDSP) are analyzed in Section 6.5. Finally, Section 6.6 addresses solution methods for the multi-vehicle problem scenario of Chapter 5.

The algorithms that we tested in our experiments were implemented in Java 1.6 and the corresponding tests were performed under Linux on a PC equipped with an Intel Core 2 Duo 3 GHz processor and 4 GB of main memory. In some test cases, solutions were obtained by solving the MIP-formulations presented in previous chapters directly via the MIP-solver of IBM ILOG CPLEX 12.1 on the same hardware. It will be clear from the following descriptions, in which cases we employed CPLEX to compute solutions.

6.1 Instance generation

This section explains in detail how we generated the problem instances that were used for the computational evaluation of the algorithms presented in this thesis. The generation process was designed to closely follow the specifications that were used by Armstrong et al. (2008), who have also conducted computational experiments to gauge the performance of their branch and bound algorithm for the described integrated production and distribution planning problem. Their evaluation is based on randomly generated test instances, which are, however, not published. For this reason we created a testbed of our own, using parameter settings similar to those in Armstrong et al. (2008).

6.1.1 Deriving instance parameters

In order to specify the global characteristics of the instances that we created, we used the following parameters as input data for the instance generator.

n	Number of customers, i.e. $\lvert C \rvert = n$
DV	Demand variability, defined as the quotient of standard deviation and mean value of the customer demands, i.e. $DV = \sigma_p / \mu_p$. This controls how large the differences between the generated demands will be on average.
$RWTW$	Relative width of time windows, defined as $RWTW := \mu_{\text{TW}} / t_S$, where μ_{TW} is the mean width of all time windows and t_S is the sum of all travel times from the plant to all customers in S and back. This parameter adjusts whether time windows will be rather narrow or wide compared to the travel time for a complete tour.
RPL	Relative product lifespan, defined as $RPL := B / t_S$. This describes whether the product has a rather short or long lifespan with respect to the time needed for transportation.
RPT	Relative production time, defined as $RPT := 1000/(R \times t_S)$. This parameter controls the fraction of time needed to produce 1000 units of the product compared with the travel time for the complete tour t_S.

For the random values of the demands and time windows we use a uniform distribution. In the following we give a more detailed description of how to generate these values and some derived parameters.

Just like Armstrong et al. we use a square of locations from $(0,0)$ to $(100,100)$ and randomly choose n (integral) locations (x_i, y_i), $i = 1, \ldots, n$, for the customers. The plant 0 is always located at the center $(50,50)$ of the square with the values p_0, a_0 and b_0 each set to zero. The travel times t_{ij} are calculated as the (rounded-up) Euclidean norm distance between the locations i and j, i.e. we set

$$t_{ij} := \left\lceil \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \right\rceil.$$

Each customer i is assigned a random, uniformly distributed demand p_i, having mean value μ_p (e.g. $\mu_p = 50$). These values can be calculated in the following way:

1. Compute the standard deviation as $\sigma_p = DV \times \mu_p$.
2. The demands will be equally distributed over an interval $I_p = [\alpha_p, \beta_p)$, whose width can be calculated using the standard deviation as $|I_p| := \sqrt{12} \times \sigma_p$
3. Choose each p_i randomly from $I_p := \left[\mu_p - \frac{|I_p|}{2}, \mu_p + \frac{|I_p|}{2}\right)$.

By computing the length of the complete tour $t_S := \sum_{i=0}^{n} t_{i,i+1}$ (where $n+1 \mathrel{\widehat{=}} 0$) we can derive the following three parameters for the problem instance:

$$\begin{aligned} \text{Mean width of time windows}: \quad \mu_{\text{TW}} &= RWTW \times t_S \\ \text{Product lifespan}: \quad B &= RPL \times t_S \\ \text{Production rate}: \quad R &= 1000/(RPT \times t_S) \end{aligned}$$

Now it remains to generate the actual time windows $[a_i, b_i]$ for each customer $i \in C$. The main idea to accomplish this consists in placing all time windows into the period between r_S^P and $(r_S^P + t_S)$, where $r_S^P := \sum_{i=1}^{n} (p_i/R)$ denotes the earliest start of transportation if the orders of all customers are selected for production. Without loss of generality we can generate the start times of the time windows in a non-decreasing manner.

If we assume a uniform distribution of the start times of the time windows, then the average distance between two consecutive start times can be set to t_S/n. We will use this value as the average increment in the start time for generating the next time window. Algorithm 6.1 gives a detailed description of the generation process for the customer time windows. It uses the values n, r_S^P, t_S, μ_{TW}, and Var_{TW} as parameters, where Var_{TW} corresponds to the demand variability and describes the quotient of standard deviation and mean value for the width (and distance) of the time windows. We use the variables D_{tw} and D_{gap} to store the resulting width of the random (uniform) distributions for the width of the time windows and the gap between consecutive start times, respectively. The helper procedure RANDOM(α, β) is

Algorithm 6.1 Generation of customer time windows

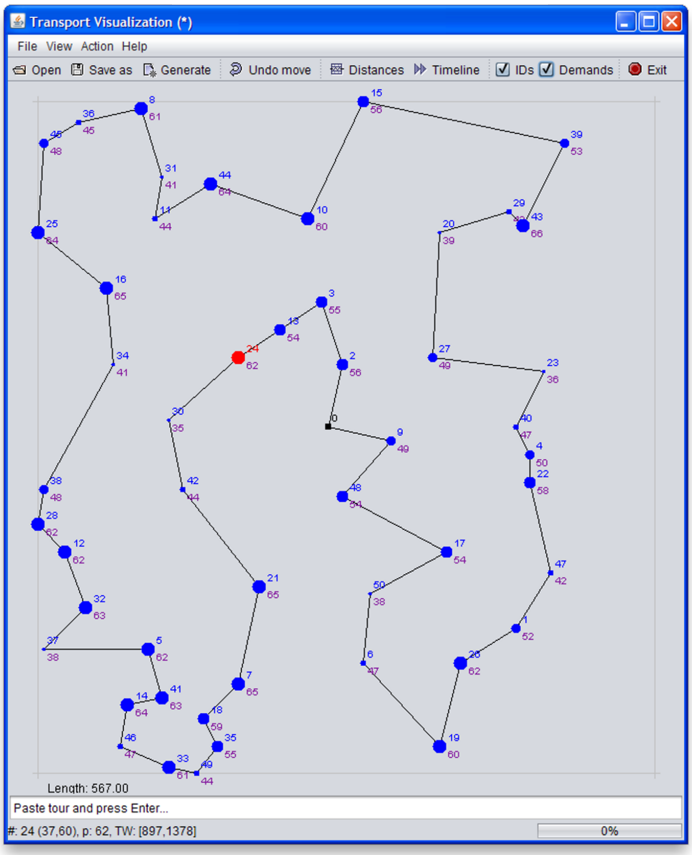
```
procedure GenerateTimeWindows(n, a_S^P, t_S, μ_TW, Var_TW)
    t ← r_S^P
    D_tw ← √12 × Var_TW × μ_TW
    D_gap ← √12 × Var_TW × (t_S/n)
    for all customers i ∈ S do
        a_i ← t
        low ← max{0, μ_TW − D_tw/2}
        twWidth ← Random(low, low + D_tw)
        b_i ← a_i + twWidth
        low ← max{1, (t_S/n) − D_gap/2}
        gap ← Random(low, low + D_gap)
        t ← t + gap
```

available (or easily implementable) in most programming languages and it is used to uniformly draw a random integer from the interval $[\alpha, \beta)$.

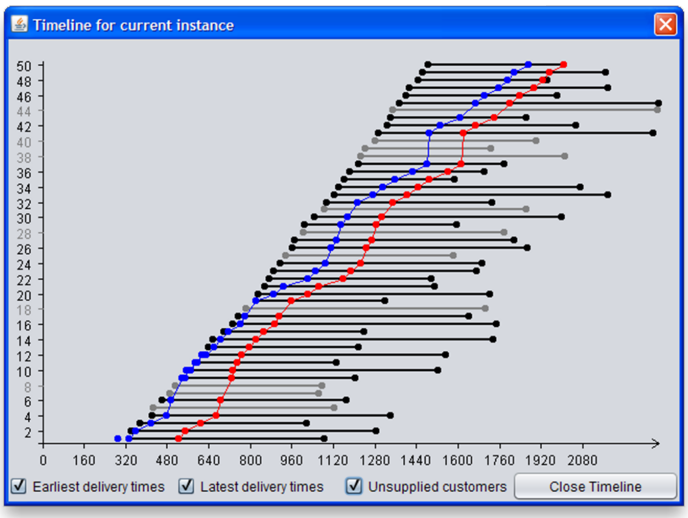
6.1.2 Travis: A tool to visualize and generate instances

Along with this thesis we have developed an application named *Travis* (*Tra*nsport *vis*ualization), whose main window is depicted in Figure 6.1(a). Since the test instances are based on locations within a (100×100)-square, they can easily be represented as shown by the blue spots in the figure. The larger the diameter of a customer location spot, the larger is its associated demand. The application window shows a tour for an example instance containing 50 customers, together with the production facility (depot) which is marked by the black square in the middle. The customers are labeled with their respective numbers (blue) and the demand (violet). The red spot indicates the currently selected customer which may be altered, e.g. by changing its parameters like the demand, time windows or location. Furthermore, the application allows to generate random instances based on the parameters mentioned above and to store them for later processing by our scheduling algorithms. Instances are saved as human-readable text files; an example is contained in Appendix B.2. Of course, such instance files can also be opened and displayed, possibly together with computed delivery tours.

As the name of the application suggests, it is also possible to compute and display transportation schedules like the one in Figure 6.1(b). This dialog presents the customer time windows (black lines for supplied customers, gray lines for unsupplied ones) and an earliest (blue) and/or latest (red) start schedule for the computed customer selection.



(a) Main application window



(b) Distribution plans: Earliest and latest start schedules for the given customer time windows

Figure 6.1: Travis: Transport visualization and instance generator

		Average percentage over 40 instances each		
n	RWTW	demand	runtime	nodes
20	0.05	1.6%	45.7%	260.2%
	0.10	1.2%	0.8%	39.9%
	0.15	0.4%	-4.0%	18.9%
	0.20	0.1%	-4.2%	17.0%
25	0.05	3.6%	243.2%	463.9%
	0.10	2.3%	34.8%	111.4%
	0.15	0.7%	17.1%	63.6%
	0.20	0.1%	12.5%	37.7%
30	0.05	7.2%	508.5%	1609.0%
	0.10	3.2%	169.5%	264.4%
	0.15	1.1%	102.7%	125.1%
	0.20	0.8%	70.8%	82.1%
35	0.05	16.7%	1366.5%	2577.2%
	0.10	5.4%	562.2%	584.0%
	0.15	1.4%	260.4%	251.2%
	0.20	0.5%	141.4%	131.7%

Table 6.1: Comparison of B&B procedures (1) using our corrected vs. (2) the incorrect bounding procedure by Armstrong et al. Percentages represent average value gains of (1) over (2), taken over 40 instances in each row. Positive percentages indicate that (1) achieved more demand, or needed more runtime or nodes, respectively.

6.2 Influence of the corrected calculation of the maximum transport delay

As shown in Section 3.1, the calculation of the maximum transport delay during the branch and bound algorithm presented by Armstrong et al. (2008) is flawed because it prunes the search tree too early in many situations where waiting times are involved. This affects the branching as well as the bounding decisions. We evaluated the impact of this error and compared our branch and bound implementations with the flawed against the corrected transportation delay calculation on a set of random test instances.

Our experiments showed that the influence of the incorrect calculation was particularly strong for instances with narrow time windows (i.e. small values of RWTW), which is

plausible because the incorrect bounding procedure is derived from slacks within the given time windows. The results are detailed in Table 6.1. For each row in the table we generated 40 random instances and compared the performances of (1) the branch and bound algorithm using our corrected bound against (2) the original algorithm. It can be observed that the ratio of explored nodes grows significantly for (1) compared to (2) for increasing values of n. Nevertheless, the achieved total satisfied demands also grow notably, where e.g. for $n = 35$ the objective function values obtained by the correct version (calculating true optima) were about 17% better than the incorrect values in method (2).

6.3 Effect of shifts for branch and bound algorithms in the basic model

In Section 3.3 we introduced delayed production starts for the model with fixed (and equal) production and distribution sequences. This model extension generally leads to an increased number of feasible schedules for a specific problem instance, i.e. the solution space becomes larger and the problem more difficult to solve. However, delaying the production start can also lead to significantly better solutions, as the results of our experiments show.

In order to evaluate the influence of allowing production delays (shifts) in the model, we generated 800 random problem instances. Our testbed consists of 40 instances for each combination of number of customers ($n = 20, 25, 30, 35$) and relative product lifespan (RPL $= 0.3, \ldots, 0.7$). We compared the performance (satisfied demand, runtime and number of nodes visited) of the B&B algorithm with shifts against that without shifts and listed the resulting percentages in Table 6.2, where positive values indicate that the algorithm with shifts achieved more demand, or needed more runtime or nodes, respectively, than the algorithm without shifts.

The results show significant differences between the two algorithms, especially for small values of RPL. In these situations, the variant with shifts was able to satisfy much more demands while at the same time needing much less computation time. This observation is especially true for larger instances, where e.g. for $n = 35$ and RPL $= 0.3$ on average about 53% more demand was satisfied by the first variant while the runtime was only about a third of the second algorithm.

For larger values of RPL, say RPL ≥ 0.6, the permission of production delays cannot justify the increased runtime of the extended B&B algorithm, since the observed satisfied demands are basically the same for both methods. However, for RPL < 0.6,

		Average percentage over 40 instances each		
n	RPL	demand	runtime	nodes
20	0.3	32.6%	18.9%	-31.4%
	0.4	10.5%	17.4%	-31.5%
	0.5	1.7%	16.8%	-12.5%
	0.6	0.1%	20.4%	-1.1%
	0.7	0.0%	18.8%	0.0%
25	0.3	34.5%	-16.9%	-41.4%
	0.4	9.9%	-8.3%	-42.5%
	0.5	0.8%	28.2%	-8.5%
	0.6	0.2%	24.6%	-4.4%
	0.7	0.0%	27.7%	0.0%
30	0.3	41.6%	-47.4%	-60.6%
	0.4	14.2%	-32.3%	-53.8%
	0.5	1.7%	11.9%	-24.7%
	0.6	0.1%	36.9%	-4.1%
	0.7	0.0%	41.0%	0.0%
35	0.3	52.8%	-64.8%	-58.9%
	0.4	15.8%	-39.6%	-56.4%
	0.5	2.4%	-9.2%	-38.6%
	0.6	0.1%	40.9%	-3.1%
	0.7	0.0%	46.4%	0.0%

Table 6.2: Comparison of B&B procedures with and without production delays (shifts) allowed: The shown values are the average percentages of satisfied demand, runtime and number of explored nodes when comparing 'B&B with shifts' against 'B&B without shifts'. Positive values indicate that the former method achieved (resp. needed) more demand (runtime, nodes) than the latter.

the B&B algorithm with shifts often explored significantly less nodes of the search tree than the version without shifts.

Table 6.3 summarizes the minimum, maximum and average runtimes needed for both branch and bound procedures on the mentioned set of instances. The rightmost column presents the ratio between the average runtimes of 'B&B with shifts' and 'B&B without shifts'. Especially for larger instances it is obvious that the relative performance depends heavily on the value of RPL, i.e. the lifespan: for short lifespans, 'B&B with shifts' solves the given instances much faster than 'B&B without shifts', which is due to the fact that shifting permits finding good solutions (with large objective values) early in the search process. This helps pruning the search tree faster. For RPL-values greater than 0.5, 'B&B with shifts' was significantly slower than 'B&B without shifts'. The reason is that shifting is not very effective when lifespan constraints become

		Mininum / Maximum / Average runtimes						
		B&B without shifts (s)			B&B with shifts (s)			Ratio
n	RPL	$t^{(1)}_{\min}$	$t^{(1)}_{\max}$	$t^{(1)}_{\mathbf{avg}}$	$t^{(2)}_{\min}$	$t^{(2)}_{\max}$	$t^{(2)}_{\mathbf{avg}}$	$t^{(2)}_{\text{avg}}/t^{(1)}_{\text{avg}}$
20	0.3	0.6	1.9	**1.1**	0.7	2.2	**1.3**	1.19
	0.4	0.5	1.5	**0.8**	0.6	1.8	**1.0**	1.19
	0.5	0.5	1.5	**0.6**	0.5	1.1	**0.7**	1.16
	0.6	0.4	0.7	**0.5**	0.5	0.9	**0.7**	1.22
	0.7	0.5	0.7	**0.5**	0.5	1.0	**0.7**	1.20
25	0.3	4.7	36.0	**17.5**	4.6	32.1	**15.4**	0.88
	0.4	1.2	17.7	**6.5**	1.6	17.8	**6.4**	0.99
	0.5	0.7	8.5	**1.9**	0.9	6.0	**2.4**	1.25
	0.6	0.5	3.3	**1.4**	0.6	4.5	**1.8**	1.27
	0.7	0.5	2.9	**1.1**	0.5	4.1	**1.4**	1.32
30	0.3	56.1	1372.4	**487.4**	25.4	1300.3	**305.1**	0.63
	0.4	21.0	1042.7	**163.8**	15.1	584.6	**112.7**	0.69
	0.5	2.0	83.2	**20.2**	2.3	109.6	**23.6**	1.17
	0.6	1.3	33.4	**6.6**	1.6	48.9	**9.4**	1.43
	0.7	1.3	26.1	**6.6**	1.8	37.6	**9.4**	1.43
35	0.3	1219.8	42711.9	**14159.5**	606.4	23529.8	**5528.0**	0.39
	0.4	260.3	13954.6	**3061.3**	184.8	6767.9	**1971.5**	0.64
	0.5	6.4	1455.2	**268.6**	6.7	1122.1	**241.6**	0.90
	0.6	3.1	309.8	**55.6**	4.2	475.7	**80.8**	1.46
	0.7	2.8	159.2	**43.6**	4.0	216.1	**64.1**	1.47

Table 6.3: Comparison of runtimes (in seconds) for branch and bound algorithms (1) without and (2) with allowed shifts of the production start. Each row contains results for a set of 40 random instances for the given number of customers and relative product lifespan (RPL).

unrestrictive. Then, both methods explore nearly the same number of nodes in the search tree, but the more computationally expensive feasibility checks for the variant with shifts lead to increased runtimes.

6.4 Evaluation of heuristic methods in the basic model

We analyzed and compared the performance of the greedy heuristic presented by Armstrong et al. (2008) and TabuSearchForSelections introduced in Section 3.4. For both algorithms delaying the production start was allowed. Tests were run on a set of 960 random test instances with 30, 80, 100, and 150 customers, respectively, where all instances shared the parameter settings for demand variability DV = 0.2 and relative production time RPT = 0.06. For each instance, 6 runs with a time limit of 10 seconds were performed. Of course, TabuSearchForSelections always used the available time completely because the time limit served as the stopping criterion for the search. In contrast, the running time of the greedy heuristic depends on the number of customers. More precisely, its complexity is $\mathcal{O}(n^4)$, which explains quickly increasing running times. In our experiments the greedy heuristic needed on average 0.2s, 1.1s, 2.0s, and 9.8s for instances with 30, 80, 100, and 150 customers, respectively.

We were particularly interested in the influence of the parameters rel. width of time windows (RWTW) and rel. product lifespan (RPL), so we conducted two test runs: in the first run we fixed RPL = 0.6 and altered RWTW between 0.05 and 0.20 (see Table 6.4(a)). In the second run, RWTW was fixed at 0.15 and RPL was altered between 0.3 and 0.6 (see Table 6.4(b)). The given tables contain the results of our experiments and show average error gaps for both heuristics, where the error gap for an instance is defined as $\frac{UB-LB}{LB} \times 100\%$. Here, UB is an upper bound on the total satisfied demand and LB represents the average objective value over 6 test runs for a particular instance. Upper bounds UB were computed by solving the instances directly via ILOG's CPLEX 12.1 with a time limit of one hour. In total, CPLEX was able to solve 900 out of 960 generated instances to optimality within that time limit, while a gap of at most 15% between the respective best found solution and the best MIP upper bound was reported by CPLEX for the remainder. For the latter instances we used the best (i.e. smallest) upper bounds that were known when the time limit was reached. The tables additionally contain the average optimality gaps reported by CPLEX for each group of instances.

Our results show that the tabu search has outperformed the deletion-based greedy heuristic in each group of test instances. The differences turned out to be particularly large for instances with very short lifespans (RPL $\leq$ 0.4), as shown in Table 6.4(b).

(a) Influence of RWTW (average values over 40 instances per row)

n	RWTW	Avg. error gap Deletion (%)	Avg. error gap Tabu search (%)	Avg. optimality gap CPLEX (%)
30	0.05	1.8	0.3	0.0
	0.10	2.6	0.1	0.0
	0.15	2.0	0.1	0.0
	0.20	1.8	0.2	0.0
80	0.05	2.4	0.1	0.1
	0.10	2.0	0.1	0.0
	0.15	2.1	0.1	0.0
	0.20	1.8	0.1	0.0
100	0.05	3.8	1.7	1.7
	0.10	2.5	0.3	0.0
	0.15	2.3	0.3	0.0
	0.20	1.8	0.2	0.0
150	0.05	10.9	9.2	9.6
	0.10	3.9	2.4	2.0
	0.15	2.4	0.9	0.2
	0.20	2.3	0.5	0.0

(b) Influence of RPL (average values over 20 instances per row)

n	RPL	Avg. error gap Deletion (%)	Avg. error gap Tabu search (%)	Avg. optimality gap CPLEX (%)
30	0.3	8.9	0.1	0.0
	0.4	4.3	0.2	0.0
	0.5	2.2	0.1	0.0
	0.6	1.7	0.2	0.0
80	0.3	48.7	0.3	0.0
	0.4	9.9	0.1	0.0
	0.5	2.0	0.1	0.0
	0.6	1.7	0.1	0.0
100	0.3	52.5	0.8	0.0
	0.4	24.7	0.4	0.0
	0.5	3.1	0.3	0.0
	0.6	1.6	0.2	0.0
150	0.3	55.0	1.7	0.0
	0.4	65.7	1.2	0.0
	0.5	52.5	0.6	0.0
	0.6	2.1	0.6	0.2

Table 6.4: Average error gaps, i.e. avg. differences between heuristic solutions and upper bounds for the deletion heuristic by Armstrong et al. and for the tabu search described in Section 3.4.

Expectedly, a clear tendency towards larger average error gaps for small values of RPL can be observed for both algorithms. Furthermore, both heuristics showed larger average error gaps when RWTW was reduced, which is evident from Table 6.4(a).

6.5 Heuristic performance for variable sequences

In this section we discuss the results of our computational experiments concerning the proposed implementations for variable production and distribution sequences. We will deal with the algorithms for the scenarios of equal (SVESP) and differing (SVDSP) sequences for production and distribution separately. However, we will also compare the results of both scenarios in order to quantify the possible gain in solution quality by allowing the sequences to differ.

6.5.1 Heuristics for the SVESP

For practical instances, the step from the basic production and distribution model in Chapter 2 to its generalization including variable sequences in Chapter 4 comes with a significant increase in problem complexity – and thus in required runtime to be solved to optimality. In order to compare the performance of the heuristics proposed to solve the SVESP (Section 4.2), we have generated a testbed of problem instances with $n = 10, 20, 30, 40, 50$ customers and four different values for RWTW (0.05, 0.10, 0.15, 0.2). For each combination of different values n and RWTW, eight random instances were used. Thus, the testbed consisted of $5 \times 4 \times 8 = 160$ instances.

n	maximum gap	average gap
10	0.0%	0.0%
20	16.0%	3.3%
30	25.6%	9.5%
40	36.1%	14.6%
50	47.6%	21.2%

Table 6.5: Optimality gaps reported by CPLEX for different problem sizes of the SVESP with a calculation time limit of one hour.

We implemented the MIP formulation of the SVESP (c.f. Section 4.2.1) in a mathematical programming language called ZIMPL[1] and used the corresponding software

[1] `http://zimpl.zib.de`, see the dissertation of Koch (2004) for details

tool to convert the data into input files that are suitable for processing with the MIP solver CPLEX 12.1. We recorded the results of running CPLEX on each instance with a time limit of 60 minutes, internally employing a parallel branch and cut (B&C) algorithm using two concurrent search threads. Within this time limit, 61 of the 160 instances could be solved to optimality (including all instances for $n = 10$), owing to the enourmous complexity of the SVESP. For the remaining instances, we obtained lower bounds (the best known solutions when the time limit was exceeded) and upper bounds (by CPLEX bounding procedures).

During the test runs the optimality gaps shown in Table 6.5 were recorded. Obviously, those gaps become large very quickly with growing numbers of customers, and sample tests on a subset of instances have shown that running the evaluation for more than 50 customers would require much larger time limits (e.g. one day per instance) to obtain suitable optimality gaps for meaningful comparisons.

Variants of SVESP heuristics under evaluation

In our tests we evaluated the performance of five different tabu search approaches for the SVESP which we will abbreviate by the following notation:

API Decomposition-based TS using the $\mathcal{N}_{api}$ neighborhood (Algo. 4.1, p. 55)

SWAP Decomposition-based TS using the $\mathcal{N}_{swap}$ neighborhood (Algo. 4.2, p. 58)

TS-SL 1 Tabu search for selection lists (Algo. 4.3, p. 60)

TS-SL 2 A variant of TS-SL 1, including randomized choices of unselected customers as insertion candidates (line 7) and selected customers as removal candidates (line 13)

TS-SL 3 A variant of TS-SL 1, where an unselected customer i to be inserted is chosen in such a way that the ratio $(b_i - a_i)/p_i^2$ is as small as possible. Customers to be removed are chosen randomly, just as in TS-SL 2.

We consider three slightly different variants of our tabu search for selection lists because in the algorithm there are several positions where customers must be chosen from a working set in some specified way. This is done in a quite greedy manner in the pseudocode presented in Chapter 4 (Algorithm 4.3), where we, for example, remove a customer with the *smallest* demand, insert another at the *best possible* position, and so on. The intuition behind the insertion criterion used in TS-SL 3 consists in the idea that those customers should be inserted first which have a narrow time window ($b_i - a_i$ is small), but a large demand (p_i^2 is large). On the one hand, by taking the square

of the demand we put an emphasis on maximizing the revenues of the production plant. On the other hand, preferring customers with narrow time windows in the insertion process will tend to leave more flexible customers for later insertions. These are typically more difficult due to a larger number of already scheduled deliveries.

Each of the five heuristics was run on the same set of problem instances as mentioned above for CPLEX. Since the solution algorithms are influenced by random decisions during their execution, each heuristic was run 6 times on each instance with a time limit of 10 seconds per run. Finally, the average objective function value over these 6 runs was taken as the result for that instance. We have deliberately chosen to use a quite short duration per test run for each instance because allowing longer computation times did not produce significantly better solutions.

Table 6.6 presents the outcome of our comparison between the heuristically obtained (average) objective function values and the corresponding results computed via CPLEX. Problem instances are grouped by the contained number of customers and by their relative widths of time windows (RWTW). Thus, each row in Table 6.6 represents results for a set of eight instances having the same parameter settings. For every heuristic, two columns are included which contain the comparisons of the average values computed by the respective heuristic and the best (smallest) upper bound found by CPLEX ("UB") within the given time limit of one hour. As already mentioned, all instances with $n = 10$ customers were solved to optimality. Consequently, the upper bounds for these instances correspond to optimal solution values.

Each entry in Table 6.6 represents an average error gap (AEG) of the heuristic solutions H compared to the corresponding upper bounds UB computed by CPLEX:

$$AEG = \left(\frac{UB - H}{H} \right) \times 100\%$$

Looking at the results, there is a clear tendency that decomposition-based heuristics (API, Swap) show better performances than the selection-list-based ones (TS-SL 1–3) for small numbers of customers, whereas TS-SL can be regarded as a better choice for $n \geq 30$.

While the Swap-based decomposition approach computes better (or at least equally good) solutions than the API variant for $n = 30$, the API-TS clearly outperforms the Swap-TS for $n \geq 40$. A reason for this is given by the fact that the more expensive exchange evaluations to search the neighborhood of the respective current solution during the Swap-TS lead to a smaller number of possible iterations within the time limit. This, in turn, causes a reduced exploration of the search space.

		Average error gaps of SVESP heuristics vs. MIP-UBs				
n	RWTW	API	Swap	TS-SL 1	TS-SL 2	TS-SL 3
10	0.05	4.2%	3.4%	**2.2**%	**2.2**%	**2.2**%
	0.10	6.8%	5.2%	**3.1**%	**3.1**%	**3.1**%
	0.15	**1.0**%	1.5%	2.1%	2.1%	2.1%
	0.20	**3.4**%	5.0%	5.0%	5.0%	5.0%
20	0.05	**0.6**%	**0.5**%	2.1%	2.1%	2.1%
	0.10	**2.5**%	**2.6**%	3.1%	3.2%	3.3%
	0.15	11.5%	11.5%	8.6%	**8.5**%	8.9%
	0.20	12.1%	11.6%	**10.1**%	10.3%	10.2%
30	0.05	**0.2**%	0.5%	0.4%	0.3%	0.6%
	0.10	19.5%	17.9%	17.0%	**16.5**%	17.3%
	0.15	13.9%	11.4%	10.6%	**10.2**%	10.4%
	0.20	10.5%	10.6%	6.2%	6.5%	**6.0**%
40	0.05	**20.7**%	22.3%	21.9%	21.8%	23.1%
	0.10	17.1%	19.8%	14.7%	**13.9**%	14.7%
	0.15	15.9%	17.8%	10.2%	**9.2**%	9.3%
	0.20	11.7%	15.7%	5.2%	**4.8**%	5.5%
50	0.05	27.2%	30.1%	26.6%	**26.1**%	29.6%
	0.10	20.2%	25.0%	15.4%	**14.8**%	16.1%
	0.15	14.3%	19.9%	8.9%	**7.8**%	9.3%
	0.20	13.2%	17.5%	4.5%	**4.1**%	5.1%

Table 6.6: Average error gaps of LS-heuristics for the SVESP compared to upper bounds computed by CPLEX with a time limit of one hour. Bold values indicate smallest (best) error gaps within each row.

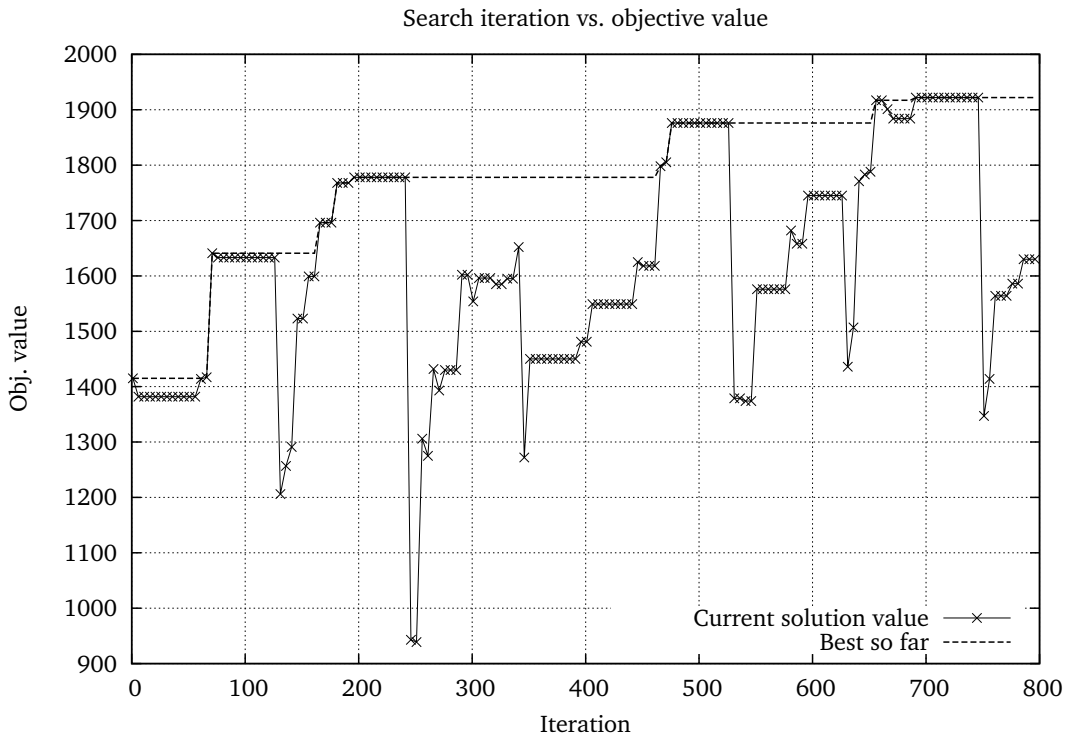


Figure 6.2: A typical test run of tabu search for selection lists for $n = 50$ customers

If we draw a comparison between the three different TS-SL approaches, we notice that their performances appear to be similar for problem instances with $n \leq 30$. Moreover, it can be observed that, beginning at $n = 40$, the TS-SL 2 variant shows slightly better results since it computes the best solutions of all five SVESP heuristics on average. This is an interesting result because TS-SL 2 represents the "most randomized" version of the implemented selection-list methods, i.e. both the insertion and removal choices in TS-SL 2 contain randomization, as opposed to TS-SL 1, where we attempt to exclude customers with the least possible demand, and TS-SL 3, where more effort is put into choosing an appropriate insertion candidate by considering differences in demand and time window flexibility.

A typical test run of tabu search for selection lists – TS-SL 2 in this case – is depicted in Figure 6.2. Here, the search was limited to perform 800 iterations, which are shown on the horizontal axis, whereas the vertical axis presents the range of observed objective function values (total satisfied demand). The dotted line indicates the best solution value found before or in the current iteration. If the search reaches a plateau (e.g. between iteration 70 and 120), where it cannot escape from the neighborhood

of a certain local optimum, the diversification step serves to leave it such that other areas of the solution space can be explored. In this example run, such plateaus are left after at most 50 iterations without improvement. Obviously, this diversification parameter plays an important role for the tabu search. On the one hand, if its value is too small, the current region of the search space will not be explored thoroughly. On the other hand, too large values will frequently leave the search stuck in the vicinity of a particular local optimum.

Comparison of SVESP heuristics to API

Until now we have analyzed the perfomance of our SVESP heuristics for rather small or medium-sized instances (10-50 customers) because suitable optimal solutions (resp. upper bounds) can only be computed within reasonable runtimes for this problem set. Nevertheless, the heuristics are also designed to be capable of solving larger instances (e.g. for more than 100 customers). Consequently, we generated an extended test set of instances which is based on the previously used smaller instances and additionally includes problems with 80, 100, 150, and 200 customers, respectively.

In our tests we used the API-based decomposition heuristic as our benchmark algorithm and compared the performance of SWAP and TS-SL 1-3 against the results of the former. The test runs were conducted in the same manner as already explained above, i.e. we took the average value of 6 runs per instance and 10 seconds per run. Since this method requires only one minute of computation time per instance, we have enlarged our test set to include 40 instances for each possible combination of number of customers and $\mathrm{RWTW} = 0.05, 0.10, 0.15, 0.20$. Accordingly, the test set consisted of 1440 instances.

The results of our comparison are shown in Table 6.7. For a heuristic method H and an instance I we computed the percentage gain in total satisfied demand compared to API as follows:

$$\left(\frac{\mathrm{Value}_H(I) - \mathrm{Value}_{\mathrm{API}}(I)}{\mathrm{Value}_{\mathrm{API}}(I)}\right) \times 100\,\%$$

The entries in Table 6.7 represent average gains over 40 problem instances. According to the definition, negative percentages indicate that API was able to find better solutions on average, whereas positive values mean better results for the respective heuristic (other than API).

For problems having between 30 and 100 customers, the SWAP heuristic (with its more expensive neighborhood explorations) yields results that are about 2–4% worse than those of the API heuristic, whereas for larger instances with 150 or even 200 customers,

		Avg. value gain compared to API			
n	RWTW	SWAP	TS-SL 1	TS-SL 2	TS-SL 3
10	0.05	1.3%	3.0%	3.0%	3.0%
	0.10	1.0%	1.9%	1.9%	1.9%
	0.15	-0.2%	-1.0%	-1.0%	-1.1%
	0.20	0.3%	0.5%	0.5%	0.5%
20	0.05	0.0%	-0.5%	-0.5%	-0.6%
	0.10	-0.3%	0.4%	0.4%	0.3%
	0.15	1.1%	3.9%	3.9%	3.7%
	0.20	1.3%	3.6%	3.7%	3.6%
30	0.05	-0.4%	-1.0%	-0.9%	-1.2%
	0.10	1.2%	2.7%	3.4%	2.9%
	0.15	1.5%	4.1%	5.0%	4.5%
	0.20	0.1%	4.0%	4.2%	3.9%
40	0.05	-2.4%	-2.1%	-2.1%	-3.6%
	0.10	-3.9%	3.9%	4.6%	3.6%
	0.15	-3.5%	5.1%	6.4%	5.8%
	0.20	-3.7%	6.6%	7.2%	6.6%
50	0.05	-2.7%	5.9%	6.8%	-1.2%
	0.10	-3.3%	11.0%	12.8%	6.8%
	0.15	-3.3%	12.7%	14.1%	10.4%
	0.20	-2.8%	13.3%	14.0%	12.3%
80	0.05	-4.4%	3.8%	5.0%	-3.0%
	0.10	-3.9%	10.1%	11.1%	6.4%
	0.15	-4.1%	11.7%	13.3%	10.6%
	0.20	-4.0%	12.8%	13.9%	11.9%
100	0.05	-1.9%	7.4%	8.4%	-1.4%
	0.10	-2.3%	13.6%	15.6%	8.0%
	0.15	-2.4%	15.3%	17.2%	11.3%
	0.20	-2.3%	13.5%	15.2%	12.1%
150	0.05	-0.2%	13.0%	14.1%	1.5%
	0.10	-0.3%	17.7%	20.0%	9.7%
	0.15	-0.3%	18.2%	20.0%	13.7%
	0.20	-0.3%	18.3%	18.9%	16.3%
200	0.05	0.1%	11.5%	12.8%	0.9%
	0.10	0.0%	16.0%	18.2%	6.0%
	0.15	0.0%	16.5%	18.7%	9.1%
	0.20	-0.1%	17.6%	18.3%	12.6%

Table 6.7: Comparison of SVESP heuristics to API for varying n and RWTW

the performances of API and SWAP are similar because both algorithms share the same method to build initial solutions. On the one hand, this is because initialization takes up a larger fraction of the allowed runtime. On the other hand, the random characteristics of the instances let both algorithms serve a similar set of customers such that the differences between satisfied demands become small compared to the overall satisfied demands for both algorithms.

The results also clearly confirm the superiority of the TS-SL heuristics over the decomposition based approaches, which is particularly evident for larger instances with $n \geq 50$, where the gains e.g. for TS-SL 2 compared to API range from 10–20%. Once again, TS-SL 2 shows the best results of all tested heuristics. An interesting observation concerns the fact that TS-SL 3 loses ground against TS-SL 1 and 2 for instances with $n \geq 50$, especially when time windows are narrow (RWTW ≤ 0.10).

The dominance of the TS-SL heuristics is made even more visible by the results in Table 6.8, which comprises the same way of comparing all implemented SVESP-heuristics against API as in the previous discussion. However, this time we have fixed the parameters RWTW $= 0.15$, RPT $= 0.06$, and DV $= 0.2$ and altered the relative product lifespan RPL between 0.3 and 0.6 for each instance size. Each row in the table presents average results over 20 instances having the stated parameters. It can be observed that the selection-lists-based algorithms produce solutions that are between 5–50% better than those generated by the API-TS, while the differences tend to be especially large for small lifespans, i.e. for RPL ≤ 0.4. The results of the SWAP heuristic are comparable to those obtained in the evaluation for varying values of RWTW, i.e. SWAP shows a similar performance that is slightly worse than API.

6.5.2 Iterated local search for the SVDSP

In this section we present the outcome of our experiments concerning the iterated local search (ILS) approach that we pursued in Section 4.3.3 to solve the SVDSP. This problem variant allows the production and distribution sequences to be different for a single vehicle.

Comparison with MIP solutions

We have formulated the SVDSP as a mixed integer program in ZIMPL (see Appendix B.1) and used it to solve a set of problem instances via CPLEX 12.1. The solving runs were interrupted after one hour of computation time and best found solutions as well as upper bounds were recorded for each instance.

		Avg. value gain compared to API			
n	RPL	SWAP	TS-SL 1	TS-SL 2	TS-SL 3
10	0.3	0.0%	1.0%	1.0%	1.0%
	0.4	0.0%	-0.6%	-0.6%	-0.6%
	0.5	0.0%	0.3%	0.3%	0.3%
	0.6	0.1%	-1.0%	-1.0%	-1.0%
20	0.3	3.6%	8.1%	8.1%	8.2%
	0.4	3.7%	8.2%	8.2%	8.3%
	0.5	1.1%	3.0%	3.3%	3.1%
	0.6	0.3%	2.4%	2.4%	2.2%
30	0.3	3.0%	12.1%	12.3%	12.1%
	0.4	2.5%	12.5%	13.3%	12.8%
	0.5	1.5%	7.4%	7.8%	7.9%
	0.6	1.1%	4.8%	5.5%	5.1%
40	0.3	1.5%	24.3%	24.6%	23.5%
	0.4	-2.7%	18.4%	18.9%	18.9%
	0.5	-4.1%	10.1%	10.4%	10.3%
	0.6	-3.8%	4.2%	5.3%	4.4%
50	0.3	-0.8%	20.3%	20.1%	18.6%
	0.4	-2.6%	23.1%	23.4%	22.9%
	0.5	-3.4%	19.2%	20.2%	11.4%
	0.6	-3.1%	12.2%	13.2%	9.9%
80	0.3	-2.7%	43.0%	43.8%	37.6%
	0.4	-3.7%	35.0%	35.5%	35.3%
	0.5	-5.1%	20.0%	20.8%	20.1%
	0.6	-4.2%	11.3%	13.1%	10.2%
100	0.3	-1.0%	48.4%	49.9%	40.1%
	0.4	-1.6%	40.5%	40.7%	40.3%
	0.5	-2.2%	26.3%	27.0%	26.7%
	0.6	-2.2%	14.5%	15.5%	10.0%
150	0.3	0.0%	13.3%	13.0%	12.9%
	0.4	-0.2%	24.5%	24.4%	24.1%
	0.5	-0.3%	24.8%	26.2%	9.1%
	0.6	-0.4%	18.2%	19.2%	13.2%
200	0.3	0.0%	13.2%	12.9%	12.7%
	0.4	0.2%	26.4%	26.3%	20.8%
	0.5	-0.1%	18.5%	20.1%	3.6%
	0.6	-0.1%	16.1%	18.4%	8.8%

Table 6.8: Comparison of SVESP heuristics to API for varying n and RPL

n	RWTW	MIP	UB	Gap
10	0.05	-15.7%	-15.7%	0.0%
	0.10	-16.8%	-16.8%	0.0%
	0.15	-4.4%	-4.4%	0.0%
	0.20	-2.6%	-2.6%	0.0%
20	0.05	-3.9%	-3.9%	0.0%
	0.10	-0.5%	-0.7%	0.2%
	0.15	0.4%	-6.5%	7.8%
	0.20	2.5%	-6.1%	9.5%
30	0.05	-2.7%	-2.7%	0.0%
	0.10	10.1%	-7.2%	20.1%
	0.15	41.4%	-3.7%	47.9%
	0.20	34.7%	-0.9%	35.9%

Table 6.9: Performances of the ILS-heuristic vs. MIP (and UB)

Since the SVDSP is extremely complex, only 59 of 96 instances in our testbed could be solved to optimality within the time limit, even though we deliberately restricted the tests to rather small instances including 10 to 30 customers. The same set of instances served to evaluate our implementation of the ILS heuristic, and we used the results of CPLEX to gauge the quality of the ILS solutions. Similar to the above mentioned test runs, we allowed 10 seconds for each ILS run and took the average satisfied demand over six such runs for each instance.

The outcome of the tests is shown in Table 6.9, where each row gives average results over eight instances. The column titled "MIP" compares the ILS results with the best known solutions (within time limit) of CPLEX, whereas the "UB" column does so for the upper bounds. Negative values in these columns indicate that the solutions of ILS were worse than the respective results of CPLEX. Of course, the percentages in the "UB" column must necessarily be non-positive. The percentages in the "Gap" column show the average optimality gaps reported by CPLEX after at most one hour.

Since optimal solutions could be computed for all of the instances with $n = 10$ and also for RWTW $= 0.05$, the values for MIP and UB do not differ in these rows. Here, the ILS heuristic was only competitive for wide time windows, i.e. RWTW ≥ 0.15. Looking at larger instances with 30 customers, ILS was able to generate up to 41% better solutions (on average) within 10 seconds than those obtained via the MIP formulation in one hour, while the ILS solutions were provably within 4% of the optimal solution values.

n	RWTW	Avg. value gain ILS vs. TS-SL2
80	0.05	11.4%
	0.10	13.6%
	0.15	8.4%
	0.20	4.8%
100	0.05	14.8%
	0.10	13.5%
	0.15	8.7%
	0.20	5.9%
150	0.05	18.6%
	0.10	12.7%
	0.15	8.4%
	0.20	5.9%
200	0.05	21.8%
	0.10	12.9%
	0.15	8.8%
	0.20	6.6%

Table 6.10: Value gains of ILS (for SVDSP) over TS-SL 2 (for SVESP)

The results allow for the interpretation that narrow time windows pose a difficulty for the ILS implementation. Notice, however, that CPLEX solved all instances with narrow time windows (RWTW $= 0.05$) to optimality. Viewed in this light, a deviation of 2.7% for $n = 30$ and RWTW $= 0.05$ is quite acceptable. Beginning at $n = 30$, instances with rather wide time windows (RWTW ≥ 0.15) turned out to be hard to solve for CPLEX, i.e. the optimality gaps remained above 30% after one hour. Further tests showed that the gaps were not significantly decreased by allowing computation times of 3 or 4 hours. However, the results of ILS prove that the MIP solver could not find an optimal solution (ILS results were better on average) and that it did not merely take the time to prove optimality.

Comparison with SVESP results

The motivation for introducing differing production and distribution sequences in the SVDSP consists in the fact that better schedules can be obtained when limited lifespans are considered (cf. Example 2 and Thm. 6 on p. 47). For this reason we have compared the results of the ILS implementation for the SVDSP to those of our

most effective SVESP heuristic, TS-SL 2, focussing on larger problem instances that contain between 80 and 200 customers. We used the same set of instances that already served as a testbed for the comparison of SVESP heuristics in Table 6.7. Hence, for each combination of instance size and usual values of RWTW we solved 40 problems. The outcome is detailed in Table 6.10, presenting percentage gains in total satisfied demand (obj. value) for ILS over the results of TS-SL 2.

The values show that the differences between best heuristic SVESP and SVDSP solutions are particularly large for narrow time windows. While for RWTW ≥ 0.1 the average gains do not seem to depend much on the instance size n, a significant increase for larger sizes can be observed for RWTW $= 0.05$, where the values range from 11% to 22%. This behavior can be explained as follows. If time windows are narrow, possible distribution sequences are severely restricted by precedence constraints implied by the time windows. In the SVESP scenario, this also implies a restricted set of production sequences. By allowing differing sequences in the SVDSP, however, the production sequence can now be adapted to make the produced orders expire as late as possible, so generally more customers can be served feasibly.

6.6 Heuristics for the multi-vehicle problem

For the MVMTP, we analyzed three variants of the ILS algorithm presented in Section 5.3 and compared their respective performances to that of the *Routing First, Production Second* (RFPS) tabu search implemented by Wendt (2009). The ILS variants differ solely in the strategy for constructing a sequence from the currently unselected customers. This procedure is called MakeListOfUnselectedCustomers in the insertion local search in Algorithm 5.2 on page 90. The variants are described in the corresponding section. In the discussion of computational results we use the following notation for the strategies and give a brief summary:

ILS_{DF}	Unselected customers are sorted by non-decreasing values $(b_i - a_i)/p_i^2$. (DF stands for demand and time window flexibility)
ILS_{TW}	Unselected customers are sorted by non-decreasing values b_i. (TW stands for end of time window)
ILS_{RND}	A random permutation is applied to the set of unselected customers. (RND stands for randomization)

In addition to that, we refer to the results of the tabu search by RFPS in the following. Each of the four heuristics was tested on a set of randomly generated instances as well

as on a set of "more structured" problems from the literature that we adapted for our setting. The outcome is contained in the following two subsections.

6.6.1 Tests on random instances

In order to evaluate heuristic algorithms for the MVMTP on random instances, an extension of our instance generation method was required. In particular, we added the number of available vehicles m and their respective capacities $q_1, \ldots, q_m$ to the parameters described in Section 6.1. Moreover, to obtain "interesting" instances we decreased the relative product lifespan RPL compared to the single-vehicle models so that multiple vehicles and/or intermediate depot returns were taken into consideration more frequently.

The tests were run on a set of 90 random instances with 50, 80, and 100 customers, respectively. For each instance size, half of the instances were generated with a homogeneous vehicle fleet (all vehicles have the same capacity), the other half with an inhomogeneous fleet (different capacities). For $n = 50$ customers we allowed $m = 2, 3, 4$ vehicles, for $n = 80$ we used $m = 3, 4, 5$, and for $n = 100$, $m = 4, 5, 6$ vehicles were considered. Furthermore, all instances shared the following parameter settings: DV = 0.2, RWTW = 0.05, RPL = 0.2. We employed the cost function (5.4) defined on page 94 and set $\alpha_1 = \alpha_2 = 0.1$ in each test run.

Table 6.11 presents the results of running each of the four mentioned heuristics with a time limit of 5 minutes per instance, which we set as a stopping criterion. The cost values obtained by RFPS were used as a benchmark. The subtables of 6.11 show different views on the results: 6.11(a) partitions the instance set by number of customers and homogeneity of the vehicles, whereas 6.11(b) contains rows for different numbers of vehicles.

While ILS_{DF} and ILS_{TW} perform poorly on instances with $n = 50$, ILS_{RND} is competitive with RFPS. For $n \geq 80$, all ILS variants beat RFPS, obtaining average cost values roughly about half the size in 6.11(a). Moreover, the gaps between the percentages for different homogeneity are significant for larger instances. The results clearly establish that ILS_{RND} shows the best average performance on the generated instances. This can be attributed to the increased diversification brought about by the randomized insertion process. In Table 6.11(b), it can be observed that the performance of ILS_{TW} and ILS_{RND} compared to RFPS is strictly (and significantly) increasing with the number of vehicles m. This is almost also true for ILS_{DF}.

(a) Comparison for varying n and homogeneous (h) or inhom. (i) vehicle fleets

			Average costs and differences to RFPS					
n	h/i	RFPS	ILS_{DF}		ILS_{TW}		ILS_{RND}	
50	h	862.3	1710.8	+98.4%	1867.8	+116.6%	878.7	+1.9%
	i	493.5	685.1	+38.8%	986.7	+99.9%	480.0	-2.7%
80	h	1729.4	984.0	-43.1%	1037.3	-40.0%	861.1	-50.2%
	i	1017.1	563.8	-44.6%	557.9	-45.2%	468.5	-53.9%
100	h	2703.8	1545.3	-42.8%	1479.3	-45.3%	1355.4	-49.9%
	i	1367.5	667.0	-51.2%	696.7	-49.1%	572.7	-58.1%

(b) Comparison for varying numbers of customers n and vehicles m

			Average costs and differences to RFPS					
n	m	RFPS	ILS_{DF}		ILS_{TW}		ILS_{RND}	
50	2	513.1	1702.6	+231.8%	1979.5	+285.8%	704.9	+37.4%
	3	674.3	1116.7	+65.6%	1464.8	+117.2%	710.2	+5.3%
	4	846.2	774.5	-8.5%	837.4	-1.0%	622.9	-26.4%
80	3	1138.2	807.0	-29.1%	760.9	-33.1%	699.0	-38.6%
	4	1354.0	679.5	-49.8%	813.4	-39.9%	645.7	-52.3%
	5	1627.6	835.2	-48.7%	818.4	-49.7%	649.8	-60.1%
100	4	1713.8	966.8	-43.6%	986.3	-42.5%	911.2	-46.8%
	5	2068.8	1228.7	-40.6%	1125.1	-45.6%	993.2	-52.0%
	6	2324.4	1123.0	-51.7%	1152.5	-50.4%	987.7	-57.5%

Table 6.11: Differences in average costs of ILS variants compared to RFPS on randomly generated instances

6.6.2 Tests on structured instances

In practice, many scheduling problems concerning production and deliveries with multiple vehicles are only inadequately characterized by random instances. For this reason, we have also analyzed the performance of our implementations on "more structured" problems, which are available for comparable problems in the literature. More specifically, we used a set of 85 test instances provided by Vansteenwegen et al. (2009b)[2] for the TOPTW. These instances are based on the problem sets of Righini and Salani (2006, 2009) and Montemanni and Gambardella (2009), who, in turn, have adapted other instances for similar problems like the VRPTW (Solomon, 1987)

[2] `http://www.mech.kuleuven.be/en/cib/op`

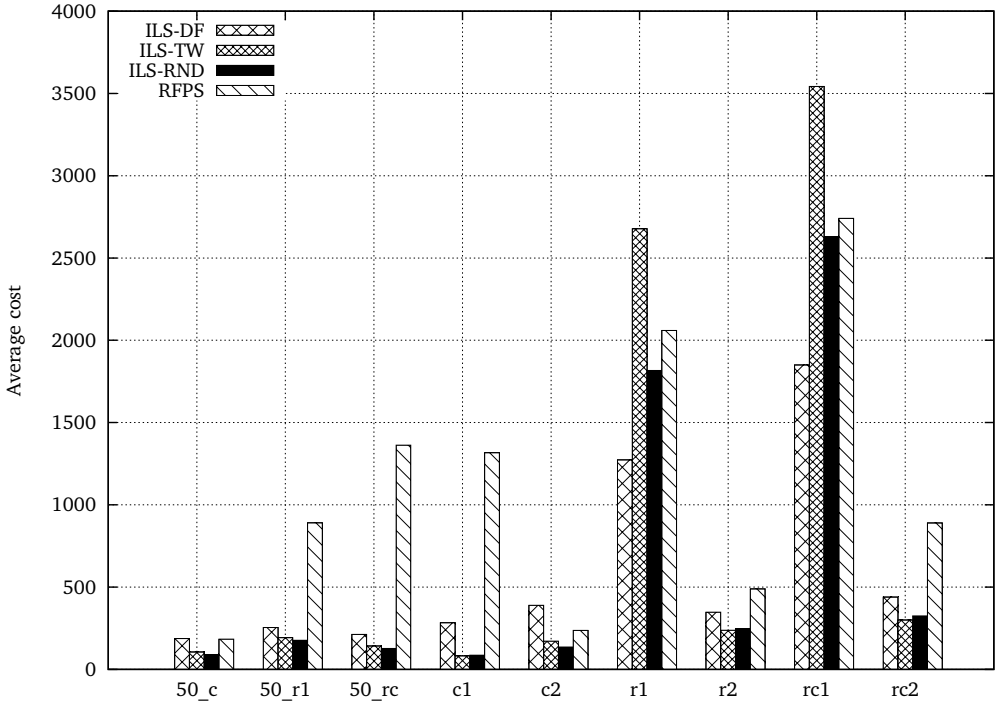
and multi-depot VRPs (Cordeau et al., 1997). Due to the additional constraints in the MVMTP compared to other problems, we had to adapt the available instances by specifying further parameters. We describe this adaptation by means of the well-known Solomon (1987) instances for the VRPTW, which are included in the testbed. These problems already contain geographical locations and time windows for customers. Furthermore, a maximum number of vehicles is given, together with corresponding vehicle capacities and a threshold on the maximum route duration, which is specified as the closing time of the depot node. We use the latter information to define the product lifespan B for the MVMTP. The Solomon instances also contain a profit information for each node, which we interpret as its demand. Since production is not considered in the VRPTW, we derive an additional value for the production rate R in the MVMTP by first computing the total length T of the tour that visits all nodes in a given instance ordered by ascending indices and then defining $R = 1000/(0.06 \times T)$. This way, the relative production time RPT is set to 6% of the total tour length (c.f. Section 6.1).

Solomon's instances can be devided into several families:

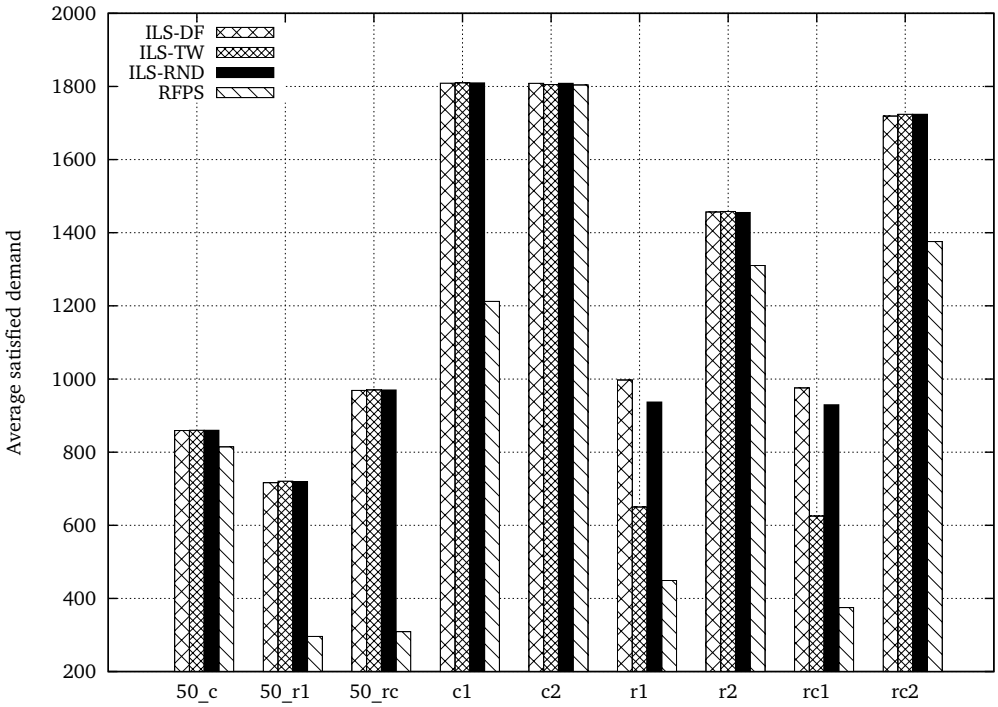
- Families c1 and c2 contain sets of geographically clustered customers, which e.g. represent different cities.
- Families r1 and r2 contain randomly generated customer locations.
- Families rc1 and rc2 may be regarded as semi-clustered, i.e. they contain a mixture of clustered and random instances.

Problem sets r1, c1, and rc1 have a short scheduling horizon, i.e. the product gets a short lifespan in our experiments for the MVMTP. This typically allows only a few customers to be served on the same tour. On the other hand, r2, c2, and rc2 have a long scheduling horizon and rather large vehicle capacities so that many customers may be served by the same vehicle. Most instances contain $n = 100$ customers; in order to analyze performances also for smaller instances, Righini and Salani (2006) created three sets of "pruned" instances including only 50 customers, which we denote by 50_c, 50_r1, and 50_rc, respectively.

We compared the performances of the three above-mentioned ILS variants to Wendt's RFPS algorithm, allowing a computation time of 5 minutes for each algorithm and instance. The results are presented as bar charts in Figure 6.3, in which the lengths of the bars indicate average costs or demands, respectively, for each of the instance families on the horizontal axis. Consequently, short bars indicate good (low-cost) solutions in the upper part (a) of the figure, whereas long bars indicate large total satisfied demands in the lower part (b).



(a) Average costs per instance family



(b) Average demands per instance family

Figure 6.3: MVMTP: Comparison of heuristics on (mostly) structured instances

	Average pctg. of served customers			
Instance set	ILS_{DF}	ILS_{TW}	ILS_{RND}	RFPS
50_c	99.8%	100.0%	100.0%	90.7%
50_r1	98.2%	99.8%	99.3%	28.7%
50_rc	99.8%	100.0%	100.0%	25.0%
c1	99.9%	100.0%	100.0%	64.0%
c2	99.9%	99.6%	99.9%	99.4%
r1	49.1%	45.1%	61.9%	23.2%
r2	99.5%	100.0%	99.7%	86.8%
rc1	39.9%	40.0%	52.6%	18.4%
rc2	99.4%	100.0%	100.0%	71.2%
average	86.7%	86.5%	90.0%	55.1%

Table 6.12: Average percentages of served customers for each instance set

It can be observed that for none of the instance families RFPS achieved the best average costs. The performance of RFPS was particularly poor for 50_r1, 50_rc, c1, and r1, where, on average, large penalty costs are incurred for RFPS due to the small number of served customers, as documented in Table 6.12. The latter also shows that ILS was able to satisfy substantially more demands than RFPS. We note that Wendt (2009) allows considerably more runtime per instance for RFPS (about 1–3 hours), so RFPS can be expected to produce better results with larger time limits. On the other hand, the ILS strategy has the advantage of producing good solutions in a short runtime, which is an observation also stated by Vansteenwegen et al. (2009b). However, tests showed that increased time limits for ILS do not yield significantly better results.

The comparison of costs in Figure 6.3(a) reveals that ILS_{RND} achieved the lowest average costs on the instance sets 50_c, 50_r1, 50_rc and c2. However, if we take the average costs and satisfied demands over all instances of all families, we obtain the following values:

	ILS_{DF}	ILS_{TW}	ILS_{RND}	RFPS
average costs	582.5	847.0	635.0	1130.7
average sat. demand	1227.9	1147.1	1216.2	853.0

Viewed in this light, ILS_{DF} shows the best performance on the set of structured test instances. Obviously, putting more emphasis on inserting customers with large demands and small time window flexibility first is more preferable than randomization in this scenario, as opposed to the success of ILS_{RND} for purely random instances.

Chapter 7

Conclusions

Integrated production and distribution scheduling has emerged as an important research area in combinatorial optimization. Due to its practical relevance on the one hand and the enormous computational complexity of the involved subproblems on the other hand, this topic of supply chain management has received an increasing amount of attention in the last few years. Several authors have shown in their computational experiments that it is indispensable to compute schedules in a coordinated manner such that production and distribution phases are well-aligned with each other. This is particularly true in make-to-order production as well as for products with a short lifespan, which we have considered in this thesis. In the following we summarize our main contributions and point out further research directions.

7.1 Summary

The research covered by this dissertation builds upon a basic production and distribution model that was introduced by Armstrong et al. (2008). After conducting an extensive review of literature related to the fields of make-to-order manufacturing, vehicle routing, orienteering problems, mathematical programming, and metaheuristics in Chapter 1, a description of the basic model and existing solution approaches is given in Chapter 2. Our decision to study extensions of this model is based on the following properties of the model:

Generalization capabilities The basic model concerns a rather restricted subproblem encountered in two-level supply chains. Due to its simplifying assumptions like fixed delivery sequences and unlimited vehicle capacities, there are many

interesting problem features that can be generalized in order to obtain models that incorporate more realistic restrictions.

Practical relevance Production and distribution scheduling problems have to be solved in practice everyday. Sample real-world applications include the joint planning of manufacturing and delivery of short-lived adhesive materials in chemical industry, ready-mix concrete for construction sites, and products like personal computers in make-to-order business scenarios.

Computational complexity Typical production and distribution scheduling problems are frequently composed of well-known computationally complex subproblems, like the $\mathcal{NP}$-hard knapsack problem (limited lifespans, vehicle capacities), routing problems like the TSP, VRP or TOP (sequencing of production and deliveries), and generalized assignment problems (orders to delivery vehicles). This combination makes the class of considered problems very challenging, both from a theoretical and a practical point of view.

A first contribution in Chapter 3 consists in the presentation of a corrected branch and bound algorithm for the basic problem. Furthermore, we have extended the model of Armstrong et al. by integrating the consideration of delayed production starts, which allows for better schedules due to the presence of lifespan restrictions. For larger problem instances we have developed an efficient implementation of the tabu search metaheuristic that can be considered a good alternative for the exact branch and bound algorithm because our computational results showed only small error gaps (within 2%) and short running times of the heuristic. The chapter concludes with a discussion of alternative objective functions, namely the minimization of travel times and the weighted delivery tardiness.

More rigorous extensions of the basic model are addressed in Chapter 4, where we discuss models for variable production and distribution sequences. The increased flexibility of these models comes at the price of growing complexity due to the addition of sequencing subproblems. We have tackled the problems with equal production and distribution sequences (SVESP) and with differing ones (SVDSP) by different heuristic solution approaches.

In order to solve the SVESP we have firstly suggested decomposition-based heuristics that separate the tasks of determining a customer sequence and a corresponding selection. This strategy was implemented within a tabu search framework. Secondly, we have implemented and analyzed TS heuristics based on a representation called selection lists, which embodies an integrated view on sequencing and selection aspects. Our experiments have confirmed that the latter approach is more effective than the decomposition-based algorithms, especially when short lifespans are involved.

Furthermore, we have proposed a solution algorithm for the scenario with differing sequences (SVDSP) that is based on iterated local search (ILS).

Chapter 5 undertakes to describe the most general integrated production and distribution models that we consider in this thesis. First, we add a capacity constraint for the transporter and allow multiple tours per workday (SVMTP) so that intermediate depot returns become necessary. Then we discuss a scenario where a fleet of transporters is responsible for the deliveries (MVMTP) while each transporter is still allowed to perform multiple tours. We propose an ILS-based solution algorithm for the MVMTP which relies on an insertion/removal neighborhood and performs the production scheduling using a branch and bound method.

In many discussions throughout this thesis we have emphasized that the introduction of lifespan constraints in all of the considered models influences the complexity of possible solution methods and prevents the direct application of established heuristics from vehicle routing literature, like those for the TSP-TW or the more general VRP-TW. Limited lifespans are the cause for the necessity to consider delayed production starts (cf. Thm. 3, p. 36), differing production and distribution sequences (cf. Thm. 6, p. 47), and the chance to construct better schedules by non-consecutive production for a single tour (cf. Lemma 9, p. 80) in the multi-vehicle case.

7.2 Outlook

The models and results presented in this thesis open a number of interesting directions for future research. In the following, we point out possibilities for extensions and alternative approaches.

Extensions For the most part, this dissertation has pursued the aim of analyzing (substantially) extended variants of the basic integrated production and distribution model presented in Section 2.1. However, there are many more conceivable ways of incorporating more realistic practical restrictions. For example, we have focused on the discussion of single-plant, single-product scenarios. Hence, an obvious generalization is to consider multiple plants that potentially produce a set of different products. Some of these scenarios may be separable by products, e.g. if they are produced on independent product lines and delivered by different transporters. In this case, models and algorithms developed in this thesis may apply. But most scenarios will require a high level of coordination among the production plant(-s) and the available transporters.

Moreover, in some problem settings it is beneficial to consider integrated schedules for multiple periods. In this case, corresponding solution algorithms are equipped with the capability of tracking the demands of each customer for several planning periods, if such data is available.

Another aspect that has been left unconsidered in our models concerns inventory decisions. On the one hand, this assumption is not too far-fetched because the goods are transported only a very short time after their production has finished. Consequently, not much space for inventories is needed at the production plant. On the other hand, inventory issues become more important if multi-level supply chains are taken into account, e.g. if further production steps at the customer's side are integrated into the scheduling models (production-transportation-production, P-T-P).

Alternative solution approaches While the test results that we obtained for our implementation of tabu search for selection lists to solve the SVESP indicate that this approach is effective and efficient, it would still be interesting to study different solution methods like other metaheuristics (e.g. genetic algorithms, simulated annealing) or even exact methods like branch and bound, which could be a viable alternative at least for smaller instances.

Furthermore, in our models it is mandatory for the scheduled deliveries to happen within the predefined customer time windows in order to obtain a feasible schedule. The local search-based algorithms that we have studied share the property that only feasible solutions are accepted in each iteration. Besides the advantage of restricting the search to the feasible region of the solution space, it could be worthwhile to analyze heuristics based on soft time windows, as already mentioned in Section 3.5.2. In this scenario, late deliveries are acceptable, but they incur penalty costs that negatively impact the objective function value.

Despite the fact that the multi-vehicle variant of the integrated production and distribution scheduling problem (MVMTP), as suggested in Section 5.2.1, features an enormous level of complexity, it is still of high practical relevance so that effective solution methods must be found that are able to cope with practical problem instances. We suggest to continue research on solution algorithms for this model, where ILS-based methods appear to be a good starting point for further investigations. Since ILS requires only short runtimes, it may also be used to obtain bounds for exact solution approaches. Moreover, Variable Neighborhood Search (VNS) as introduced by Hansen and Mladenovic (2001) has been shown to represent an effective approach to solve rich variants of VRPs (Paraskevopoulos et al., 2008). Thus, an adaption of VNS for solving the MVMTP could be worth considering.

Appendix A

Notation

A.1 Symbol reference

n	Total number of customers/orders (1 order per customer)
$C = \{1, \ldots, n\}$	Set of customers/orders (one order per customer)
$N = C \cup \{0, n+1\}$	Nodes of a transport graph, including dummy customers 0 and $n+1$ for the depot
A	Arcs of a transport graph, connecting central depot and customers
$S = (S_1, \ldots, S_n)$	Customer sequence $S \in C^n$, where $S_i \neq S_j$ for all $i \neq j$
$\sigma = (\sigma_1, \ldots, \sigma_k)$	Selection of customers/orders to be supplied, $\sigma \subseteq S$
B	Product lifespan
R	Constant production rate of the production facility
p_i	Demand of customer $i \in C$
$p(\sigma)$	Demand satisfied for a feasible selection σ, $p(\sigma) = \sum_{i \in \sigma} p_i$
$[a_i, b_i]$	Time window for the delivery of order $i \in C$
t_{ij}	Travel time from customer $i \in C$ to customer $j \in C$, location 0 (or $n+1$) represents a special location, denoting the production facility

t_S	Sum of all travel times for a tour from the facility to all customers in S and back (without consideration of time windows or lifespans), i.e. $t_S = \sum_{i=0}^{n} t_{i,i+1}$, where $0 \hat{=} n+1$
τ_{σ_i}	Delivery time of order $\sigma_i \in \sigma$, $a_{\sigma_i} \leq \tau_{\sigma_i} \leq b_{\sigma_i}$
w_{σ_i}	Waiting time at customer $\sigma_i \in \sigma$
W_{σ_i}	Cumulative waiting time before τ_{σ_i}, $W_{\sigma_i} := \sum_{j=1}^{i} w_{\sigma_j}$
c_{σ_i}	Production completion time of order σ_i
c_S	Production completion time, if all orders in S are processed
Δ_{σ_i}	Delivery tardiness of order σ_i
l_{σ_i}	Lifespan tardiness of order $\sigma_i \in \sigma$
L_σ	Maximum lifespan tardiness
δ_{σ_i}	Delaying potential of order σ_i, cf. Def. 2 on p. 35
δ_σ	Delaying potential of the first order in σ, i.e. $\delta_\sigma = \delta_{\sigma_1}$
$ESS_\sigma(t)$	The earliest start schedule for selection σ, when the production is delayed by t time units
$\mathcal{F}_\sigma$	The feasibility time window for the start of production of a given selection σ (cf. Thm. 5 on p. 37)

Multiple tours and/or vehicles

$K = \{1, \ldots, k\}$	Set of possible tours (multi-tour/-vehicle scenario)
$V = \{1, \ldots, m\}$	Set of vehicles (multi-vehicle scenario)
q_v	Capacity of vehicle $v \in V$
$[r_i, d_i]$	Production time window for order $i \in C$
ρ	A route in a multi-vehicle schedule

A.2 Glossary

3PL	Third-party logistics (provider)
AEG	Average Error Gap
ESS	Earliest Start Schedule
GRASP	Greedy randomized adaptive search procedure
ILS	Iterated Local Search
LS	Local Search
LSS	Latest Start Schedule
MIP	Mixed Integer Linear Program (-ming)
MVMTP	Multiple Vehicles, Multiple Tours, Variable Sequences Problem (5.2.1, p. 77)
$\mathcal{NP}$	The complexity class of problems solvable in polynomial time by a *non-deterministic* Turing machine (NTM)
OMS	One-machine Sequencing Problem
OP(-TW)	Orienteering Problem (with Time Windows)
$\mathcal{P}$	The complexity class of problems solvable in polynomial time by a *deterministic* Turing machine (DTM)
SCM	Supply chain management
SVDSP	Single Vehicle, Differing Sequences Problem (4.3, p. 60)
SVESP	Single Vehicle, Equal Sequences Problem (4.2, p. 49)
SVMTP	Single Vehicle, Multiple Tours, Variable Sequence Problem (5.1, p. 72)
TOP(-TW)	Team Orienteering Problem (with Time Windows)
TS	Tabu Search
TSP(-TW)	Traveling Salesman Problem (with Time Windows)
VMI	Vendor-managed Inventory Replenishment
VNS	Variable Neighborhood Search
VRP(-TW)	Vehicle Routing Problem (with Time Windows)

Appendix B

Formulating MIPs in ZIMPL

B.1 Formulation for the SVDSP

This section presents an implementation of the MIP model we have developed for the single-vehicle, differing sequences problem (SVDSP) as introduced in Section 4.3. In our tests we used a modeling language called ZIMPL which was developed by Koch (2004) at the Konrad-Zuse-Zentrum Berlin[1].

ZIMPL provides a command line program that translates linear programming models to input files like `*.lp` or `*.mps` which are better suited to be processed by MIP solvers like CPLEX[2], Gurobi[3], or SCIP[4], to name a few.

The ZIMPL-file extracts the required parameters from an instance file like the one presented afterwards in Appendix B.2.

```
1  ##################### MIP file for SVDSP ##################################
2  param datafile := "instance.txt";
3
4  # Read parameters from datafile
5  param n := read datafile as "2n" comment "#" use 1;
6  param V := read datafile as "2n" comment "#" skip 1 use 1;
7  param s := read datafile as "2n" comment "#" skip 2 use 1;
8  param B := read datafile as "2n" comment "#" skip 3 use 1;
9  param R := read datafile as "2n" comment "#" skip 4 use 1;
```

[1] http://zimpl.zib.de
[2] http://www.ibm.com/software/integration/optimization/cplex
[3] http://www.gurobi.com
[4] http://scip.zib.de

```
# Index sets
set Customers := { 1..n };
set Nodes     := Customers + {0, n+1};

# Read further parameters (depending on set 'Nodes')
param p[Nodes] := read datafile as "<2n>␣3n" comment "#" skip 5;
param a[Nodes] := read datafile as "<2n>␣4n" comment "#" skip 5;
param b[Nodes] := read datafile as "<2n>␣5n" comment "#" skip 5;
param x[Nodes] := read datafile as "<2n>␣6n" comment "#" skip 5;
param y[Nodes] := read datafile as "<2n>␣7n" comment "#" skip 5;

# Production times
param prodTime[<j> in Nodes] :=
        if (j==0 or j==n+1) then 0 else ceil(p[j]/R) end;

# Euclidean distance functions
defnumb dist(P,Q) := ceil(sqrt(P * P + Q * Q));
defnumb travel(i,j) :=  if (j==n+1)
                        then dist(x[0] - x[i], y[0] - y[i])
                        else dist(x[j] - x[i], y[j] - y[i]) end;

defnumb end(j)   := if (j==n+1) then b[0] else b[j] end;
defnumb begin(j) := if (j==n+1) then a[0] else a[j] end;

# Arc sets
set PossibleArcs := {<i,j> in Nodes*Nodes with i!=j and i <= n and j>=1};
set Arcs := { <i,j> in Nodes*Nodes with i != j and i <= n and j >= 1
              and begin(i) + travel(i,j) <= end(j) };

# Big-Ms
param M_p := (sum <i> in Customers: p[i]) + (max <i> in Customers: p[i]);
param M_t := (max <i> in Customers: b[i]) + (max <i,j> in Arcs: travel(i,j));

############################## Variables ################################

# Production sequence
var X[PossibleArcs] binary;

# Delivery sequence
var Y[Arcs] binary;

# Production finishing times
var C[Nodes] implicit integer;

# Delivery times
var TAU[Nodes] implicit integer;

# Selected flag for customers
var S[Customers] binary;
```

```
########################## Objective function ############################

# Maximize the total satisfied demand
maximize obj: sum <i> in Customers: S[i] * p[i];

######################### Restrictions ##################################

# Flow conservation
subto flowConservY: forall <i> in Customers:
    sum <h,i> in Arcs: Y[h,i] == sum <i,j> in Arcs: Y[i,j];
subto flowConservStartY:
    sum <0,j> in Arcs: Y[0,j] == 1;
subto flowConservEndY:
    sum <i,n+1> in Arcs: Y[i,n+1] == 1;

subto flowConservX: forall <i> in Customers:
    sum <h,i> in Arcs: X[h,i] == sum <i,j> in Arcs: X[i,j];
subto flowConservStartX:
    sum <0,j> in Arcs: X[0,j] == 1;
subto flowConservEndX:
    sum <i,n+1> in Arcs: X[i,n+1] == 1;

subto productionTime: forall <i,j> in Arcs:
    C[i] + prodTime[j] - M_p * (1 - X[i,j]) <= C[j];

subto travelingTime: forall <i,j> in Arcs:
    TAU[i] + travel(i,j) - M_t * (1 - Y[i,j]) <= TAU[j];

subto produced_i: forall <j> in Customers:
    sum <i,j> in Arcs: X[i,j] == S[j];

subto S: forall <j> in Customers:
    sum <i,j> in Arcs: Y[i,j] == S[j];

subto endOfProductionEqualsStartOfTransport:
    C[n+1] == TAU[0];

subto timeWindowStart: forall <i> in Customers:
    a[i] * S[i] <= TAU[i];

subto timeWindowEnd:   forall <i> in Customers:
    b[i] * S[i] >= TAU[i];

subto lifespan: forall <i> in Customers:
    TAU[i] - C[i] <= B;
```

B.2 An example input file

The following text file illustrates the input format for the problem instances that we generated for testing. Lines containing only whitespace or beginning with '#' are ignored as comments. Other entries list the

- number of nodes (n)
- number of vehicles (V)
- vehicle capacities (s)
- product lifespan (B)
- prodution rate (R)
- customer-specific data (c)

Here is an example including eight customers and two delivery vehicles:

```
######################## Example Instance ################################
# Nr of customers
n 8

# Nr of vehicles
V 2

# List of vehicle capacities / sizes (sep. by whitespace)
s 200 250

# Product lifespan
B 310

# Production rate
R 20.0

# Customer (Nr, demand, start TW, end TW, xPos, yPos)
c 0 0 0 1000 50 50
c 1 45 20 47 79 61
c 2 49 77 96 56 87
c 3 49 161 184 10 35
c 4 56 197 226 22 2
c 5 49 275 305 5 82
c 6 37 330 350 5 96
c 7 56 370 405 78 89
c 8 44 408 437 93 59
```

Bibliography

Archetti, C., Hertz, A., and Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76.

Armstrong, R., Gao, S., and Lei, L. (2008). A zero-inventory production and distribution problem with a fixed customer sequence. *Annals of Operations Research*, 159(1):395–414.

Awerbuch, B., Azar, Y., Blum, A., and Vempala, S. (1999). New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2007). An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, 178(3):755–766.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763.

Bard, J. and Nananukul, N. (2009). The integrated production-inventory-distribution-routing problem. *Journal of Scheduling*, 12(3):257–280.

Boudia, M. and Prins, C. (2009). A memetic algorithm with dynamic population management for an integrated production-distribution problem. *European Journal of Operational Research*, 195(3):703–715.

Boussier, S., Feillet, D., and Gendreau, M. (2007). An exact algorithm for team orienteering problems. *4OR: A Quarterly Journal of Operations Research*, 5(3):211–230.

Bräysy, O. and Gendreau, M. (2002). Tabu search heuristics for the vehicle routing problem with time windows. *TOP*, 10(2):211–237.

Brucker, P. and Knust, S. (2006). *Complex Scheduling*. GOR-Publications. Springer, Berlin.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47.

Chandra, P. and Fisher, M. L. (1994). Coordination of production and distribution planning. *European Journal of Operational Research*, 72(3):503–517.

Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996). The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474.

Chen, Z.-L. (2010). Integrated Production and Outbound Distribution Scheduling: Review and Extensions. *Operations Research*, 58(1):130–148.

Chen, Z.-L. and Pundoor, G. (2009). Integrated order scheduling and packing. *Production and Operations Management*, 18(6):672–692.

Chen, Z.-L. and Vairaktarakis, G. L. (2005). Integrated Scheduling of Production and Distribution Operations. *Management Science*, 51(4):614–628.

Chvatal, V. (1983). *Linear Programming*. W. H. Freeman and Company, New York–San Francisco.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Cordeau, J.-F., Gendreau, M., and Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936.

Cornillier, F., Laporte, G., Boctor, F. F., and Renaud, J. (2009). The petrol station replenishment problem with time windows. *Computers & Operations Research*, 36(3):919–935.

Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1):80–91.

Denning, P. J. (2005). The locality principle. *Communications of the ACM*, 48(7):19–24.

Dueck, G. and Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161–175.

Federgruen, A., Prastacos, G., and Zipkin, P. H. (1986). An allocation and distribution model for perishable products. *Operations Research*, 34(1):75–82.

Federgruen, A. and Zipkin, P. (1984). A combined vehicle routing and inventory allocation problem. *Operations Research*, 32(5):1019–1037.

Feillet, D., Dejax, P., and Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205.

Garcia, J. and Lozano, S. (2004). Production and vehicle scheduling for ready-mix operations. *Computers & Industrial Engineering*, 46(4):803–816.

Garcia, J. M. and Lozano, S. (2005). Production and delivery scheduling problem with time windows. *Computers & Industrial Engineering*, 48(4):733–742.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

Geismar, H. N., Laporte, G., Lei, L., and Sriskandarajah, C. (2008). The integrated production and transportation scheduling problem for a product with a short lifespan. *INFORMS Journal on Computing*, 20(1):21–33.

Gendreau, M., Laporte, G., and Semet, F. (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2-3):539–545.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

Glover, F. and Laguna, M. (1993). Tabu search. In Reeves, C. R., editor, *Modern heuristic techniques for combinatorial problems*, chapter 3, pages 70–150. Blackwell, Oxford, UK.

Glover, F. and Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326.

Hansen, P. and Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

Hertz, A. and de Werra, D. (1990). The tabu search metaheuristic: How we used it. *Annals of Mathematics and Artificial Intelligence*, 1:111–121.

Hurter, A. P. and Van Buer, M. G. (1996). The newspaper production/distribution problem. *Journal of Business Logistics*, 17(1):85–107.

Kantor, M. G. and Rosenwein, M. B. (1992). The Orienteering Problem with Time Windows. *The Journal of the Operational Research Society*, 43(6):629–635.

Koch, T. (2004). *Rapid Mathematical Programming*. PhD thesis, Technische Universität Berlin.

Laporte, G. and Martello, S. (1990). The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193–207.

Lei, L., Chaovalitwongse, W., and Bora, S. (2007). Scheduling the operations of an integrated production-distribution process. In Baptiste, P., Kendall, G., Munier-Kordon, A., and Sourd, F., editors, *Proc. of the 3rd Multidisciplinary Int. Conf. on Scheduling: Theory and Applications (MISTA)*, pages 316–327, Paris, France.

Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Lourenço, H., Martin, O., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G. A., editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers.

Melo, R. A. and Wolsey, L. A. (2010). Optimizing production and transportation in a commit-to-delivery business mode. *European Journal of Operational Research*, 203(3):614–618.

Millar, H. H. and Kiragu, M. (1997). A time-based formulation and upper bounding scheme for the selective travelling salesperson problem. *The Journal of the Operational Research Society*, 48(5):511–518.

Montemanni, R. and Gambardella, L. M. (2009). Ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 34(4):287–306.

Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, New Jersey.

Paraskevopoulos, D., Repoussis, P., Tarantilis, C., Ioannou, G., and Prastacos, G. (2008). A reactive variable neighborhood tabu search for the heterogeneous fleet vehicle routing problem with time windows. *Journal of Heuristics*, 14(5):425–455.

Potts, C. N. (1980). Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28(6):1436–1441.

Ramesh, R., Yoon, Y.-S., and Karwan, M. H. (1992). An Optimal Algorithm for the Orienteering Tour Problem. *INFORMS Journal on Computing*, 4(2):155–165.

Resende, M. G. C. and Ribeiro, C. C. (2005). GRASP with path-relinking: Recent advances and applications. In Ibaraki, T., Nonobe, K., and Yagiura, M., editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer.

Righini, G. and Salani, M. (2006). Dynamic programming for the orienteering problem with time windows. *Note del Polo-Ricerca*, 91.

Righini, G. and Salani, M. (2009). Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203.

Sakai, S., Togasaki, M., and Yamazaki, K. (2003). A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2-3):313–322.

Sarmiento, A. M. and Nagi, R. (1999). A review of integrated analysis of production–distribution systems. *IIE Transactions*, 31(11):1061–1074.

Savelsbergh, M. W. P. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305.

Schrage, L. (1970). Solving resource-constrained network problems by implicit enumeration–nonpreemptive case. *Operations Research*, 18(2):263–278.

Schrage, L. (1971). Obtaining optimal solutions to resource constrained network scheduling problems. Unpublished manuscript, cited after Potts (1980).

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Souffriau, W. (2010). *Automated Tourist Decision Support*. PhD thesis, Katholieke Universiteit Leuven.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., and Van Oudheusden, D. (2010). A path relinking approach for the team orienteering problem. *Computers & Operations Research*, 37(11):1853–1859.

Stecke, K. E. and Zhao, X. (2007). Production and Transportation Integration for a Make-to-Order Manufacturing Company with a Commit-to-Delivery Business Mode. *Manufacturing Service Operations Management*, 9(2):206–224.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Tang, H. and Miller-Hooks, E. (2005). A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32(6):1379–1407.

Tarantilis, C. D. and Kiranoudis, C. T. (2001). A meta-heuristic algorithm for the efficient distribution of perishable foods. *Journal of Food Engineering*, 50(1):1–9.

Toth, P. and Vigo, D., editors (2002). *The vehicle routing problem*. SIAM Monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia.

Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *The Journal of the Operational Research Society*, 35(9):797–809.

Tsirimpas, P., Tatarakis, A., Minis, I., and Kyriakidis, E. G. (2008). Single vehicle routing with a predefined customer sequence and multiple depot returns. *European Journal of Operational Research*, 187(2):483–495.

Van Buer, M. G., Woodruff, D. L., and Olson, R. T. (1999). Solving the medium newspaper production/distribution problem. *European Journal of Operational Research*, 115(2):237–253.

Vanderbei, R. J. (2008). *Linear Programming, Foundations and Extensions*, volume 114 of *International Series in Operations Research & Management Science*. Springer, New York, 3 edition.

Vansteenwegen, P., Souffriau, W., and Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D. (2009a). A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118–127.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D. (2009b). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D. (2009c). Metaheuristics for tourist trip planning. In Geiger, MJ et al., editor, *Metaheuristics in the Service Industry*, volume 624 of *Lecture Notes in Economics and Mathematical Systems*, pages 15–31. Springer Berlin Heidelberg.

Viergutz, C. and Knust, S. (2010). Integrated production and distribution scheduling with lifespan constraints. working paper (submitted).

Wendt, R. (2009). Integrierte Produktions- und Transportplanung mit mehreren Fahrzeugen. Master's thesis, University of Osnabrück.

Zhong, W., Chen, Z.-L., and Chen, M. (2010). Integrated production and distribution scheduling with committed delivery dates. *Operations Research Letters*, 38(2):133–138.