Two-Stage Vehicle Routing Problems with Profits and Buffers

Analysis and Metaheuristic Optimization Algorithms

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet

Informatik

vorgelegt

von M. Sc. Hoang Thanh Le

geboren am 21.10.1993 in Köthen (Anhalt)

Die Annahme der Dissertation wurde empfohlen von:

1. Professor Dr. Martin Middendorf, Universität Leipzig
2. Professor Dr. Thomas Weise, Hefei University, China

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 01.06.2023 mit dem Gesamtprädikat *summa cum laude*.

Abstract

This thesis considers the Two-Stage Vehicle Routing Problem (VRP) with Profits and Buffers, which generalizes various optimization problems that are relevant for practical applications, such as the Two-Machine Flow Shop with Buffers and the Orienteering Problem. Two optimization problems are considered for the Two-Stage VRP with Profits and Buffers, namely the minimization of total time while respecting a profit constraint and the maximization of total profit under a budget constraint. The former generalizes the makespan minimization problem for the Two-Machine Flow Shop with Buffers, whereas the latter is comparable to the problem of maximizing score in the Orienteering Problem.

For the three problems, a theoretical analysis is performed regarding computational complexity, existence of optimal permutation schedules (where all vehicles traverse the same nodes in the same order) and potential gaps in attainable solution quality between permutation schedules and non-permutation schedules. The obtained theoretical results are visualized in a table that gives an overview of various subproblems belonging to the Two-Stage VRP with Profits and Buffers, their theoretical properties and how they are connected.

For the Two-Machine Flow Shop with Buffers and the Orienteering Problem, two metaheuristics 2BF-ILS and VNS_{OP} are presented that obtain favorable results in computational experiments when compared to other state-of-the-art algorithms. For the Two-Stage VRP with Profits and Buffers, an algorithmic framework for Iterative Search Algorithms with Variable Neighborhoods (ISAVaN) is proposed that generalizes aspects from 2BF-ILS as well as VNS_{OP}. Various algorithms derived from that framework are evaluated in an experimental study. The evaluation methodology used for all computational experiments in this thesis takes the performance during the run time into account and demonstrates that algorithms for structurally different problems, which are encompassed by the Two-Stage VRP with Profits and Buffers, can be evaluated with similar methods.

The results show that the most suitable choice for the components in these algorithms is dependent on the properties of the problem and the considered evaluation criteria. However, a number of similarities to algorithms that perform well for the Two-Machine Flow Shop with Buffers and the Orienteering Problem can be identified. The framework unifies these characteristics, providing a spectrum of algorithms that can be adapted to the specifics of the considered Vehicle Routing Problem.

Acknowledgments

First and foremost, I want to express my gratitude to my supervisor, Prof. Dr. Martin Middendorf, for his guidance and advice throughout the years, for the inspiring discussions with him and for providing an environment that allows me to focus on research.

Many thanks to my wonderful colleagues from the Swarm Intelligence and Complex Systems Group, to Nicolas Wieseke for helping me with the various troubles I had over the years, to Özhan Kayacan and Fatma Turna for their advice and for always listening to my worries, to all the people from the research group who I first got to know as fellow "Doktoranden", some of whom I even saw completing their doctoral degree: Thanks to Tobias Jagla, Deisy Gysi, Tom Hartmann, Felix Brei, Lisa Fiedler, Dominik Vietinghoff, Daniel Abitz, Stefan Preußner and Jan Witte. The conversations I had with you showed me that I was not alone with my struggles and reminded me not to get too absorbed in my research bubble.

I want to express my gratitude to Bernd Krause for his help and advice during my time as an undergraduate student and for teaching me many things that I still use to this day.

A special thanks all of my friends. Whenever I spend time with you, I can take a break away from research and have fun while still learning new things outside my area of research.

Also, I want to thank my family. Thanks to my brother Ngoc Le Dang for always lending an ear when I had problems and for all the laughs we had about the most trivial of things. And finally, many thanks to my parents Nguyen Thi Mai and Chau Le Dang for their continued support throughout my life. Without you, I would not be a part of this world; I would not be in the here and now. Thank you.

Contents

1 Introduction **9**

2 Background **13**

2.1 Problem Motivation 13

2.2 Formal Definition of the Two-Stage VRP with Profits and Buffers . . . 16

2.3 Review of Literature on Related Vehicle Routing Problems 23

2.3.1 Two-Stage Vehicle Routing Problems 24

2.3.2 Vehicle Routing Problems with Profits 26

2.3.3 Vehicle Routing Problems with Capacity- or Resource-based Restrictions 28

2.4 Preliminary Remarks on Subsequent Chapters 31

3 The Two-Machine Flow Shop Problem with Buffers **35**

3.1 Review of Literature on Flow Shop Problems with Buffers 36

3.1.1 Algorithms and Metaheuristics for Flow Shops with Buffers . 36

3.1.2 Two-Machine Flow Shop Problems with Buffers 39

3.1.3 Blocking Flow Shops 40

3.1.4 Non-Permutation Schedules 42

3.1.5 Other Extensions and Variations of Flow Shop Problems . . . 43

3.2 Theoretical Properties 44

3.2.1 Computational Complexity 44

3.2.2 The Existence of Optimal Permutation Schedules 57

3.2.3 The Gap Between Permutation Schedules an Non-Permutation Schedules 58

3.3 A Modification of the NEH Heuristic 72

3.4 An Iterated Local Search for the Two-Machine Flow Shop Problem with Buffers 74

3.5 Computational Evaluation . . . 77
3.5.1 Algorithms for Comparison . . . 77
3.5.2 Generation of Problem Instances . . . 77
3.5.3 Parameter Values . . . 78
3.5.4 Comparison of 2BF-ILS with other Metaheuristics . . . 79
3.5.5 Comparison of 2BF-OPT with NEH . . . 86
3.6 Summary . . . 87

4 The Orienteering Problem 89
4.1 Review of Literature on Orienteering Problems . . . 92
4.2 Theoretical Properties . . . 96
4.3 A Variable Neighborhood Search for the Orienteering Problem . . . 102
4.4 Computational Evaluation . . . 106
4.4.1 Measurement of Algorithm Performance . . . 106
4.4.2 Choice of Algorithms for Comparison . . . 107
4.4.3 Problem Instances . . . 109
4.4.4 Parameter Values . . . 110
4.4.5 Experimental Setup . . . 110
4.4.6 Comparison of VNS_{OP} with other Metaheuristics . . . 111
4.5 Summary . . . 116

5 The Two-Stage Vehicle Routing Problem with Profits and Buffers 119
5.1 Theoretical Properties of the Two-Stage VRP with Profits and Buffers 119
5.1.1 Computational Complexity of the General Problem . . . 120
5.1.2 Existence of Permutation Schedules in the Set of Optimal Solutions . . . 120
5.1.3 The Gap Between Permutation Schedules an Non-Permutation Schedules . . . 126
5.1.4 Remarks on Restricted Cases . . . 135
5.1.5 Overview of Theoretical Results . . . 156
5.2 A Metaheuristic Framework for the Two-Stage VRP with Profits and Buffers . . . 164
5.3 Experimental Results . . . 173
5.3.1 Problem Instances . . . 173

5.3.2 Experimental Results for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ 175
5.3.3 Experimental Results for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ 181
5.4 Summary 186

Bibliography **189**

List of Figures **211**

List of Tables **215**

List of Algorithms **217**

1 Introduction

The term "Vehicle Routing Problem" refers to a variety of combinatorial optimization problems that occur in many practical applications and play an important role in the area of operations research. The general idea common to all of these problems is that routes need to be planned for a set of vehicles such that certain criteria (so-called "optimization criteria") are maximized or minimized.

There exist a variety of performance indicators that can be potentially used as optimization criteria, such as the travel time, route length, travel cost, number of used vehicles or vehicle emissions as examples for optimization criteria to be minimized, or the number of visited customers, the profit from customers or the coverage of an area by vehicles as examples for maximization criteria. Furthermore, it is possible that various restrictions (so-called "constraints") are imposed on the problem that need to be taken into account, for example limited capacities for vehicles, length restrictions for routes or time windows that limit the time when customers are available. This shows that Vehicle Routing Problems can occur in a variety of forms and with a multitude of additional conditions. Due to the breadth of this research area, it is not uncommon that review articles for Vehicle Routing Problems only focus on selected aspects, such as certain types of vehicles (for example, [175]), vehicle routing problems with certain types of constraints (two examples being [121, 3]) or routing in a specific application (for example, [154]).

This thesis considers a type of Vehicle Routing Problem with exactly two vehicles where the locations that are visited by the vehicles have tasks (which are also called "jobs"). The work on these tasks is done by these two vehicles in two stages with each stage taking a certain amount of time. Fully processing both stages yields a profit, but it is not possible to freely process jobs as potential time limits or budget constraints limit the number of jobs that can be processed. Or alternatively, it is possible that routes need to be calculated that take the least amount of time while collecting a minimum profit. In addition, there is a "buffer constraint" that needs

to be taken into account, which, intuitively speaking, states that there cannot be too many jobs that have only finished one of the two processing stages. This is interpreted in the sense that jobs in that state take up space in a "buffer" with limited capacity.

In the following, this type of Vehicle Routing Problem is referred to as the **Two-Stage Vehicle Routing Problem with Profits and Buffers** (or short, Two-Stage VRP with Profits and Buffers). At first glance, this problem might appear to be an arbitrarily restricted problem with additional special conditions, but later in this thesis it is shown that it encompasses various optimization problems that are actively researched in the literature and important for practical applications, such as the Orienteering Problem or even the Two-Machine Flow Shop with Buffers, the latter being a scheduling problem that is not directly connected to vehicle routing. For this reason, investigating this Vehicle Routing Problem in this thesis not only aims to improve understanding of the general problem, but also provide new insights into other problems contained in the Two-Stage VRP with Profits and Buffers.

To this end, theoretical and empirical analyses are performed in this thesis. On the theoretical side, the analysis focuses on several aspects that can be informally stated as follows:

- Is this optimization problem a hard or an easy problem?
- Is it possible to calculate "optimal" solutions when the routes for all vehicles are identical?
- If that is not the case, how much "quality" is lost when all vehicles are required to traverse the same route?

For the two aforementioned special cases of the problem, which are encompassed by the Two-Stage VRP with Profits and Buffers and relevant for practical applications, two algorithms 2BF-ILS and VNS_{OP} are presented in this thesis and empirically evaluated. Since the Two-Stage VRP with Profits and Buffers contains other problems, a framework for metaheuristics is proposed from which various algorithms can be derived. The so-called "ISAVaN" framework generalizes aspects of 2BF-ILS and VNS_{OP} and can be used to fit an algorithm to the specifics of the problem. The heuristics developed for the special cases as well as the algorithms for

the general Two-Stage VRP with Profits and Buffers are evaluated using a consistent methodology in order to demonstrate that algorithms developed for structurally different problems (that are actually encompassed by the Two-Stage VRP with Profits and Buffers) can be analyzed using similar methods. This methodology can also be easily extended due to its general formulation for optimization algorithms of this type. In short, this thesis aims to contribute new insights into theoretical aspects as well as new heuristic and empirical methods for this type of Vehicle Routing Problem.

Parts of this thesis are based on published research of the author, in particular [93, 57, 96, 97]. However, all computational experiments from these works were redone on a larger scale and their evaluation has been extended to cover additional aspects. Their results are restructured in a coherent framework, so that this thesis also aims to illustrate the relationships between these works and the research problems considered in them.

The following chapters of this thesis are structured as follows. An introduction to the Two-Stage VRP with Profits and Buffers is given in Chapter 2 where the problem is formally described and an overview of related Vehicle Routing Problems is presented. Chapter 3 and Chapter 4 deal with two important special cases of the problem that are relevant for practical applications and actively researched in the literature, namely the Two-Machine Flow Shop with Buffer and the Orienteering Problem. Both of these chapters have a similar structure: First, an overview of research literature for these problems is given, after which theoretical properties are analyzed. The analysis focuses on the aspects outlined in the informal questions above which are further specified in the following chapters. Afterwards, heuristic algorithms are presented for these problems and evaluated in experimental studies.

Finally, Chapter 5 considers the general case of the Two-Stage VRP with Profits and Buffers. In that chapter, the theoretical analysis is further expanded to consider additional related aspects, after which the findings are presented in a systematic overview to illustrate how the theoretical results obtained in this thesis are connected. Furthermore, the aforementioned metaheuristic framework for the Two-Stage VRP with Profits and Buffers is presented and various algorithms derived from this framework are compared in an experimental study.

2 Background

In this chapter, a background on the Two-Stage VRP with Profits and Buffers is given. Afterwards, the problem is formally defined and a literature overview regarding related Vehicle Routing Problems is presented.

2.1 Problem Motivation

In order to gain an intuitive understanding of the problem, it is reasonable to first provide an informal description. As described in the previous chapter, in the "Two-Stage VRP with Profits and Buffers" there are 2 vehicles for which routes need to be planned. On these routes, locations (or "customers") are visited that are serviced in two stages (or "steps"), which each stage (as well as the routes between customers) taking a certain amount of time. The servicing done by the two vehicles is split between them so that one vehicle can only do the servicing for the first stage and the other vehicle can only do the servicing for the second stage. It is possible that both vehicles visit the locations in a different order, but the second servicing stage at a given location can only be started after the first one has been completed. In addition, it is assumed that the service for the second stage always takes the same amount of time at each location.

Completing both stages at a location yields a profit, so it is desirable to collect a high profit without having the routes for each vehicle take too much time. More precisely, two problems can be informally stated based on this description:

1. Collect as much profit as possible within a given time budget.

2. Obtain a certain minimum profit in the shortest time possible.

In addition, there is a constraint that is referred to as the "buffer constraint" in the following. It states that at any point in time there cannot be too many

customers where only the first servicing stage is done, but not the second. This can be interpreted in the sense that such customers occupy "space" in a buffer (similar to a "waiting list" or "waiting space") with limited capacity and it is not allowed that the buffer is overfilled.

As mentioned previously, this Vehicle Routing Problem contains various conditions and constraints that appear to be arbitrarily chosen, in particular with respect to the buffer and the constant servicing times for the second stage. However, various interpretations are conceivable to illustrate how these concepts can potentially occur in an application.

For example, the buffer constraint can also be interpreted as a limited resource that is shared by both vehicles (or their drivers). This means that servicing a location (or a customer) during the first stage takes up resources which are regained or replenished when the service for the second stage is performed, and some services might have to be delayed if insufficient resources are available. In that sense the buffer constraint shares aspects from resource allocation problems. Examples for limited resources of this type can be found in computer networks, where computers (that correspond to "vehicles" in the Vehicle Routing Problem) perform calculations or process jobs. In this context, the buffer constraint can refer to a shared memory between computers or shared computing power on a server cluster that is accessed by clients.

Another interpretation is from the perspective of financial credit. In this case, the service for the first stage incurs a certain cost or debt that is repaid at a later time during the second stage. In this sense, occupying the buffer can be seen as overdrawing resources and the buffer constraint means that the overdraft cannot exceed a certain value.

Regarding the restriction that the service for the second stage takes the same amount of time, it is conceivable in a variety of applications that the second stage only consists of a routine task that approximately takes the same amount of time (e.g., routine checks or maintenance services), whereas the service done during the first stage is specific to the customer or location taking a variable amount of time (e.g., repair services, consultation or medical treatment). Other examples for routine tasks can be the packaging of manufactured products, customer billing, clean-ups or the loading and unloading of objects from the vehicle. Based on these examples, various application scenarios are conceivable where the Two-Stage VRP

with Profits and Buffers plays an important role. Some scenarios are described in the following to gain an intuitive understanding of the considered Vehicle Routing Problem.

Apart from the obvious scenario with "customers" being visited by "vehicles" where the second stage consists of routine work not specific to the customer, another scenario is connected to the aforementioned computer network. In this case, the vehicles correspond to two computers with a shared memory (which acts as the "buffer"). In this example, the first "stage" can be the query of a database and saving the query results, whereas the second "stage" can be a routine task, such as the addition of metadata and moving the data to another location outside of the memory (to free buffer space). The "times to travel between locations" in the original formulation can correspond to the loading and unloading of different databases for the queries and the "profits" can be used to represent the importance of queries or the payment for their processing.

Alternatively, when using the aforementioned example with computing power, the client responsible for the first stage (the "first processing step") allocates resources from a server cluster to perform computing tasks, whereas another computer checks the validity of the results and deallocates these resources (the second stage) so that they can be used for different tasks. In this scenario, it is not hard to see the the buffer with limited capacity corresponds to the server cluster where it is not allowed that more resources are allocated than available. The "travel times between customers" can correspond to loading or preparation times for these tasks, whereas profit values can be used to represent the monetary value or the importance of a computing task.

Another scenario is conceivable where the first stage corresponds to a vehicle or a driver performing jobs at a chemical plant where these jobs also lead to waste products. These unwanted byproducts need to be disposed of by a second vehicle before the amount of waste products exceeds a limit set by regulations. This regulation corresponds to the buffer constraint, the travel times to the driving times between locations and the profits can represent the priority of jobs, the payment for these jobs or the financial compensation for the disposal of waste products.

Practical applications can also be found in special cases that are contained in the Two-Stage VRP with Profits and Buffers. One such example is the Two-Machine Flow Shop with Buffers due to its structural similarities (which are further described

below, see Chapter 3). This problem originally comes from the area of scheduling, but it also has applications in other areas, such as the streaming of multimedia files [86, 88, 107, 108]. In this scenario, the "customers" are the media files, the two stages are represented by the preloading (also called "prefetching") and the playback of media files, respectively, and the buffer corresponds to a limited storage on a computer. This can also be extended to incorporate times for loading and unloading different media sources (which form the "times to travel between locations") and scores which describe the priority or the cost for downloading media files, in which case this scenario would also be related to the general Two-Stage VRP with Profits and Buffers.

The examples given above demonstrate that a variety of practical scenarios are conceivable for the Vehicle Routing Problem considered in this thesis, including some that are not directly related to the routing of actual "vehicles". For this reason, research on this problem is relevant not only for academic reasons, but also for practical applications. Furthermore, these examples illustrate different forms of the Two-Stage VRP with Profits and Buffers in order to give an intuitive understanding of this problem which is formally described in the following.

2.2 Formal Definition of the Two-Stage VRP with Profits and Buffers

An instance of the Two-Stage VRP with Profits and Buffers is characterized by several problem components. First, the basic terms are introduced.

- A directed, complete and simple **graph** $G = (V, E)$ is given with nodes $V = \{v_0, v_1, \ldots, v_n\}$ where v_0 is the **start node** (or "depot node").

- Assigned to every non-depot node v_i with $i \in \{1, 2, \ldots, n\}$ is a **job** J_{v_i} (or "task, customer") that is processed in two steps (or "**two stages**"). For the first stage, a time $a_{J_{v_i}} \geq 0$ is needed, whereas for the second stage $b_{J_{v_i}} \geq 0$ time units are required. As described above, the special case is studied where for all J_{v_i} it holds that $b_{J_{v_i}} = c$ with a non-negative constant $c \geq 0$. If the node v_i underlying the job J_{v_i} is clear from the context, the abbreviations $J_i = J_{v_i}$, $a_{J_i} = a_{J_{v_i}}$, $b_{J_i} = b_{J_{v_i}}$ are also used. In the following, the values a_{J_i}, b_{J_i} are referred

to as **processing times** (or "service times"). The set of all jobs $\{J_1, J_2, \ldots, J_n\}$ is denoted $\mathcal{J}$. Note that graph G contains $n+1$ nodes, whereas the number of jobs is n since the depot node v_0 has no job.

- Two vehicles M_1, M_2 are given which traverse G starting from the depot node v_0. The notation M_1, M_2 is based on the notion of "machines" or "workers" in scheduling problems that process jobs or tasks. In the context of the problem considered in this thesis, M_1 and M_2 can be considered to be "movable machines" (or **machines** for short) that traverse the graph and process jobs J_i. More precisely, vehicle M_1 is only responsible for the first processing step of a job J_i (which takes a_{J_i} time units), whereas M_2 is only responsible for the second processing step of J_i (which takes b_{J_i} time units). For this reason, the values a_{J_i} and b_{J_i} can be interpreted as "the processing time of job J_i on M_1 and M_2", respectively.

- For the processing of jobs J_i by M_1 and M_2, the following **rules** must be followed:
  - A machine M_k ($k \in \{1,2\}$) needs to be at the node v_i in order to start the processing of the job $J_i = J_{v_i}$. A machine cannot move while it is processing a job.
  - It is not allowed for one of the machines M_1, M_2 to process a job J_i multiple times. The processing of a job by a machine cannot be interrupted once it has started (i.e., the processing is non-preemptive).
  - For a job J_i the second processing step (the servicing in the second stage) can only start if its first processing step has already been completed. In other words, machine M_2 can only start processing a job J_i if M_1 has already finished the processing of J_i.

- In addition, every job J_i yields a **profit** $r_{J_i} > 0$ (or "score", "reward"). The profit r_{J_i} of a job J_i is only considered to be "obtained" once both processing steps of J_i are completed. Due to the rules for processing jobs described above, it is equivalent to say that the profit is obtained when M_2 finishes the processing of J_i.

- The edges $e_{ij} = (v_i, v_j)$ of G are weighted with non-negative values $d_{i,j} \geq 0$

where $d_{i,j}$ corresponds to the **travel time**, i.e., the time needed to traverse the edge e_{ij}. Note that the travel time $d_{i,j}$ is independent from the machine that traverses the edge e_{ij}.

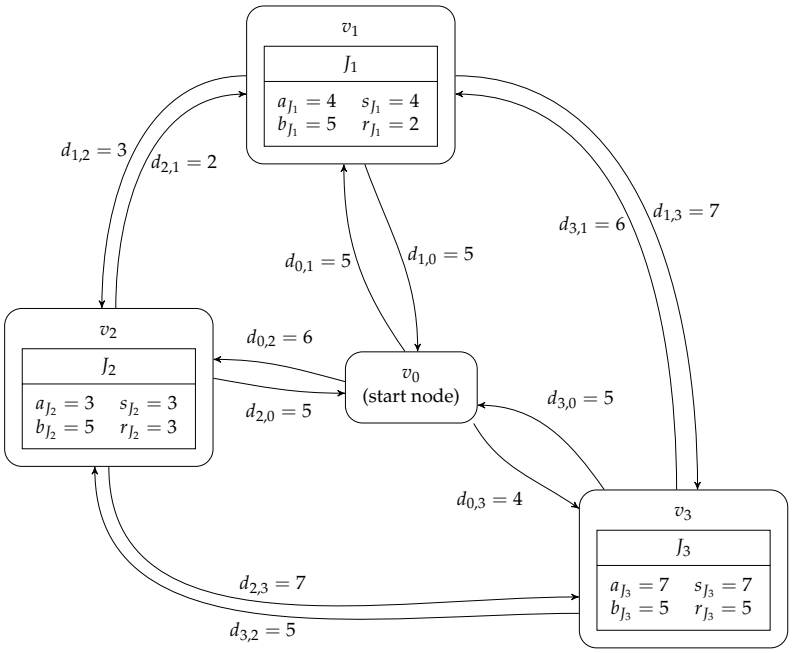


Figure 2.1: Visualization of an example graph with jobs for the Two-Stage VRP with Profits and Buffers. The jobs assigned to each node are shown inside the nodes, along with their processing times a_J, b_J on M_1 and M_2 as well as the profits r_J obtained for processing them. The notation "s_J" refers to the amount of buffer space occupied by a job J. The labels on the directed edges indicate their travel times.

An example graph for the Two-Stage VRP with Profits and Buffers with one depot node, three non-depot nodes and three jobs is shown in Figure 2.1. Next, some terms are introduced to describe the movement of vehicles M_1, M_2 in the graph.

- A **schedule** σ specifies a pair of paths $(P^1(\sigma), P^2(\sigma))$ through nodes of G (starting at the depot node v_0) describing the sequence in which M_1 and M_2 visit nodes v and process the corresponding jobs J_v. For a path $P^k(\sigma) = (v_0, v_1^{(k)}, v_2^{(k)}, \ldots, v_{\ell_k}^{(k)})$ with $k \in \{1,2\}$, it is required that $\ell_1 = \ell_2 = \ell$ (i.e., both paths contain the same number $\ell \leq n$ of nodes) and that each node

in $P^k(\sigma)$ occurs at most once in $P^k(\sigma)$, i.e., it is not possible for M_k to visit the same node v and process the corresponding job J_v twice. Furthermore, the sets of nodes visited by M_1 and M_2 whose jobs are processed must be equal: $\{v_0, v_1^{(1)}, v_2^{(1)}, \ldots, v_\ell^{(1)}\} = \{v_0, v_1^{(2)}, v_2^{(2)}, \ldots, v_\ell^{(2)}\}$. This set is denoted *VisitedNodes*(σ) in the following. It is not required for M_1 or M_2 to return to the depot node at the end.

- The sequence $\pi^k(\sigma)$ of processed jobs corresponding to a path $P^k(\sigma)$ is defined as $\pi^k(\sigma) = (J_{v_1^{(k)}}, J_{v_2^{(k)}}, \ldots, J_{v_\ell^{(k)}})$. Similarly, it is required that the sets of jobs processed by M_1 and M_2 are equal:

$$\{J_{v_1^{(1)}}, J_{v_2^{(1)}}, \ldots, J_{v_\ell^{(1)}}\} = \{J_{v_1^{(2)}}, J_{v_2^{(2)}}, \ldots, J_{v_\ell^{(2)}}\} \subseteq \mathcal{J}$$

  Note that it is possible that $\ell \leq n$, which means that only a subset of the available jobs is processed in σ. The set of processed jobs is denoted *ProcessedJobs*(σ).

- In a schedule σ, the time when M_k starts the processing of J_i is denoted by $S^k_{J_i}(\sigma)$ (**starting time**). If the considered schedule σ is understood from the context, the abbreviation $S^k_{J_i}$ is used.

- The time when M_k finishes the processing of J_i (**completion time**) is denoted by $C^k_{J_i}(\sigma)$. For M_1, the completion time for a job J_i is equal to $C^1_{J_i}(\sigma) = S^1_{J_i} + a_{J_i}$, whereas for M_2 it can be calculated using the formula $C^1_{J_i}(\sigma) = S^2_{J_i} + b_{J_i}$. The notation $C^k_{J_i}$ is also used in the following if the considered schedule σ is clear from the context.

- Using this notation, the condition that M_2 can only process jobs which have been completed on M_1, can be expressed as $S^2_{J_i}(\sigma) \geq C^1_{J_i}(\sigma)$. Note that $S^2_{J_i}(\sigma) = C^1_{J_i}(\sigma)$ is allowed which means that M_2 immediately starts the processing of J_i after J_i has been finished on M_1.

- A schedule σ is **valid** if the starting times and completion times specified by σ satisfy the rules described above for the processing of jobs.

- A schedule σ is a **permutation schedule** if the paths $P^1(\sigma), P^2(\sigma)$ traversed by M_1 and M_2 through the graph G are equal, i.e., they visit the same nodes

in the same order so that the notation $P(\sigma)$ can be used instead of $P^1(\sigma)$ or $P^2(\sigma)$. This also implies that the sequences $\pi^1(\sigma)$, $\pi^2(\sigma)$ of jobs processed by M_1 and M_2 are equal, in which case the notation $\pi(\sigma)$ is used to refer to the order of jobs on M_1 and M_2.

In order to describe the constraints of this problem, the notion of a buffer is introduced. As outlined in the examples presented in the previous section, the buffer can correspond to various concepts, such as a shared pool of resources with limited capacity, computational resources (e.g., a shared memory or a server cluster), regulations imposed on byproducts generated during the processing of jobs, or even overdraft limits on bank accounts, depending on the considered application.

- There is a buffer with limited **capacity** Ω which is occupied by jobs J_i and the amount of buffer space taken by J_i is denoted s_{J_i}. Regarding the duration in which a job takes up space in the buffer, two buffer models are considered.

- In the **intermediate buffer** model, a job J_i takes up buffer space when it finishes its processing on M_1 without immediately starting its processing on M_2. It leaves the buffer when M_2 starts to process J_i. Formally, buffer space is occupied from $C^1_{J_i}$ until $S^2_{J_i}$. If M_1 finishes a job J_i while M_2 is not ready to process J_i and the free capacity in the buffer is smaller than s_{J_i}, machine M_1 is **blocked**, i.e., the processing is finished, but M_1 cannot move to another node or process other jobs until M_2 starts to process J_i or until enough buffer space is freed to store J_i. In the following, this is indicated by the parameter $\mathsf{bufType}$ ("buffer type") having the value $\mathsf{bufType} = \textit{intermediateBuffer}$.

- In the **spanning buffer** model, a job J_i takes up buffer space during the entirety of its processing, i.e., from $S^1_{J_i}$ until $C^2_{J_i}$. If M_1 arrives at a node v_i where starting the processing of the corresponding job J_i would lead to the buffer capacity being exceeded (and where J_i is the next job to be processed in the schedule), M_1 is forced to wait until enough buffer space becomes available to store J_i. The corresponding value for the parameter $\mathsf{bufType}$ is $\mathsf{bufType} = \textit{spanningBuffer}$.

- For the values s_{J_i}, two choices are common in the literature regarding scheduling problems with buffers: One method is to interpret the buffer as a "counter" for the number of stored jobs where Ω is the maximum number of jobs that

can be stored at a time. This corresponds to setting $s_{J_i} = 1$ for all jobs. This case is investigated in the scheduling literature, e.g., by [191, 200] and [96] and indicated by the buffer usage parameter bufUsage having the value "$s_J = 1$".

The second method for defining s_{J_i} is to set this value equal to the processing times on the first machine, i.e., $s_{J_i} = a_{J_i}$ for all jobs J_i. The intuition behind this method is that the "size" of the job J_i is interpreted as being proportional to the "amount of work" during its first processing stage. Examples for this choice in scheduling problems are found in [50, 88, 86] and [107]. Using the parameter bufUsage, this case is indicated by the value "$s_J = a_J$".

- A schedule σ is **feasible** if it is valid and if the buffer capacity is not exceeded at any time. In the following, it is assumed that only feasible schedules are considered, so that in the following the term "schedule" refers to feasible schedules, unless noted otherwise.

- For schedules σ it is assumed in the following that all jobs are processed "as early as possible", i.e., if it is possible for M_k ($k \in \{1, 2\}$) to start the processing of a job J_i without violating the buffer constraint (and if for M_k the job J_i is the next job to be processed according to σ), then it is assumed that M_k immediately starts the processing of J_i.

An example solution with the visualization of the processing times and the buffer usage is shown in Figure 2.2. Using the notation introduced above, the schedule σ depicted in that figure specifies the paths $P^1(\sigma) = (v_0, v_1, v_2, v_3)$ and $P^2(\sigma) = (v_0, v_2, v_1, v_3)$ and the sequence of processed jobs is $\pi^1(\sigma) = (J_1, J_2, J_3)$ and $\pi^2(\sigma) = (J_2, J_1, J_3)$ for M_1 and M_2, respectively. Note the time intervals in which buffer space is occupied for the two different buffer models. In the case where a spanning buffer is used (bottom right in Figure 2.2), note how M_1 has to wait before processing J_3 or else the buffer capacity would not exceeded. The jobs processed in σ are $\mathit{ProcessedJobs}(\sigma) = \{J_1, J_2, J_3\}$, i.e., all available jobs are processed in this example.

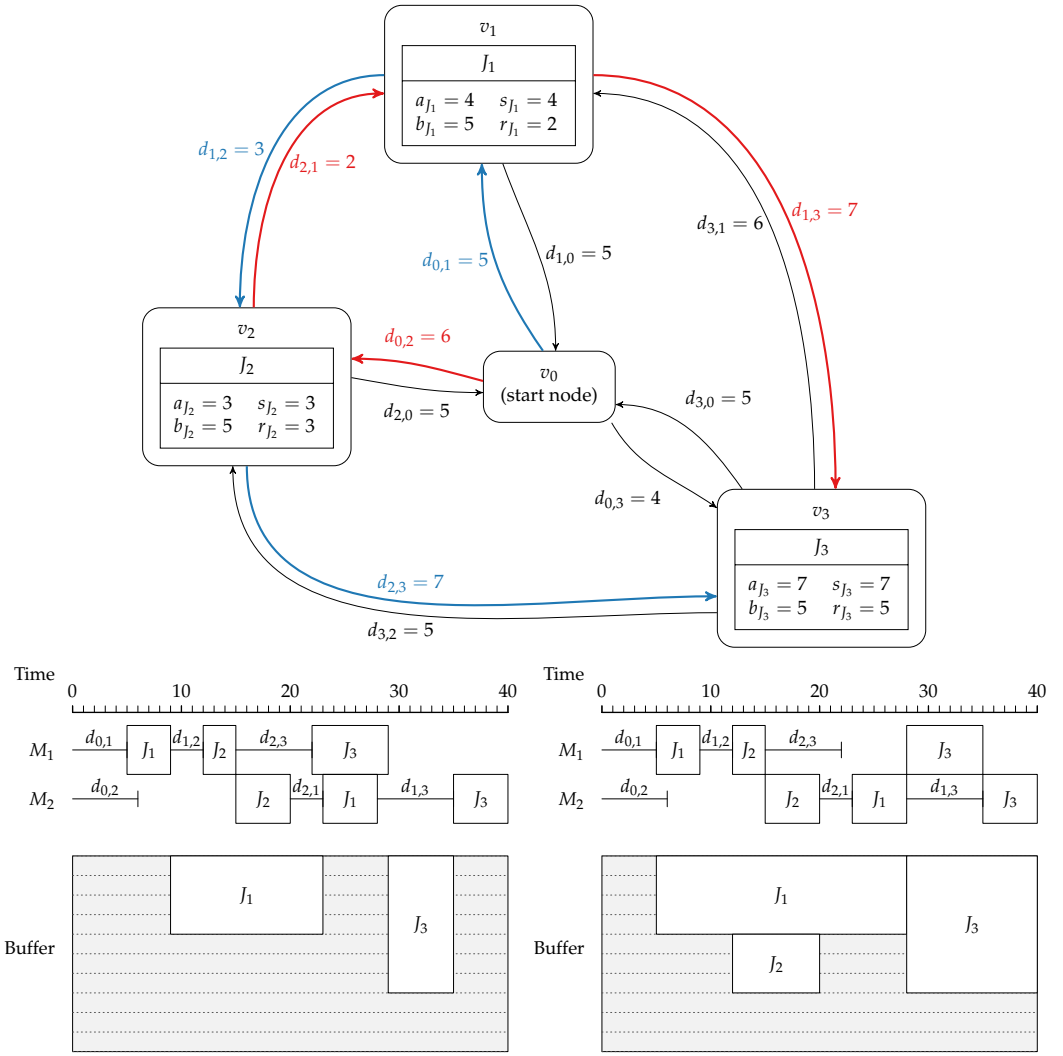


Figure 2.2: Top: Visualization of an example route for the instance of the Two-Stage VRP with Profits and Buffers from Figure 2.1. The blue edges are the edges traversed by M_1, whereas the red edges indicate the edges traversed by M_2. Bottom left: The resulting schedule for the depicted route in the case of an intermediate buffer with capacity $\Omega = 10$. Bottom right: Resulting schedule for the same route when a spanning buffer with capacity $\Omega = 10$ is used.

Finally, some terms regarding the optimization criterion are introduced.

- Given a schedule σ, the **total length** $C_{\max}(\sigma)$ (or **makespan**) of σ is the time when M_2 finishes the processing of the last job, i.e., $C_{\max}(\sigma) = \max_J C_J^2(\sigma)$, where the maximum is taken over all jobs J in $\mathit{ProcessedJobs}(\sigma)$. Regarding the example schedule σ shown in Figure 2.2, its makespan is $C_{\max}(\sigma) = 39$ for both buffer models.

- The **total profit** $R(\sigma)$ (or "total score", "total reward", "total value") of a schedule σ is the sum over all rewards r_{J_i} belonging to the jobs J_i processed in σ. It is defined as $R(\sigma) = \sum r_J$, where the sum is taken over all jobs J in $\mathit{ProcessedJobs}(\sigma)$. In the example schedule σ shown in Figure 2.2, the total profit of σ is $R(\sigma) = 10$.

- Using the terms above, two optimization problems are considered in this thesis:
  - $\mathcal{O}_{\max R,\, C_{\max} \leq B}$: Given a **budget** B, find a schedule σ with $C_{\max}(\sigma) \leq B$ (**budget constraint**) that maximizes the total profit $R(\sigma)$.
  - $\mathcal{O}_{\min C_{\max},\, R \geq Q}$: Given a **minimum score** Q, find a schedule σ with $R(\sigma) \geq Q$ (**minimum score constraint**) that minimizes the makespan $C_{\max}(\sigma)$.

Note that even though these optimization problems have different target criteria, both entail the problem of selecting a suitable subset of nodes (and jobs) as well as the problem of calculating a short route through the selected nodes. The concepts and notation introduced above are used throughout this thesis.

2.3 Review of Literature on Related Vehicle Routing Problems

There already exists a vast amount of literature regarding vehicle routing problems and their variants. In this section, an overview of literature on vehicle routing problems related to the Two-Stage VRP with Profits and Buffers is given. This overview is divided into multiple sections where in each section a different aspect of the Two-Stage VRP with Profits and Buffers is considered for which related works are presented.

2.3.1 Two-Stage Vehicle Routing Problems

An interesting type of routing problem occurs when it consists of two layers that interact with each other. This property usually occurs when a vehicle routing problem is combined with additional aspects that arise in practical scenarios. One example is a two-stage problem with time windows for arcs considered by Çetinkaya, Karaoglan, and Gökçen [30]. The first stage deals with a routing problem from a "facility" to multiple depots, whereas the second stage is to route vehicles from the depots to the customers in a tripartite graph. Each stage has its own vehicle fleet and the time windows on the arcs cause the two stages to affect each other. This problem is inspired from military applications where some of the roads are only safe during daytime, but it can also be applied to routing in public transportation where certain roads should be avoided during rush hours. They present a mixed-integer linear program and propose a Memetic Algorithm for this problem.

A similar problem is investigated in [133] where the freight transportation is to be planned (i) from an origin point to one of multiple depots (that have limited capacity) and (ii) from a depot to a customer using vehicles with limited capacity. While it is possible that each vehicle in the first stage transports freight designated for multiple customers, it is not possible for customers in the second stage to receive freight from multiple vehicles. In addition, each depot has its own cost factor and costs are incurred for each freight unit delivered to that depot. The authors propose math-heuristics for the cost-minimization problem and present inequalities allowing for cuts in the corresponding mixed linear program.

An interesting application is presented by Shen, Dessouky, and Ordóñez [155] who consider a routing problem in the scenario of a bio-terrorism emergency. In their work, the first stage is the "planning stage" where routes are planned in advance based on stochastic information. The second stage is the "operational stage" where the emergency occurs, the stochastic information is replaced by concrete values and the previously calculated routes are to be adapted. The problem is to minimize the unmet demand, and a Tabu Search is proposed as well as multiple approximation heuristics for the second stage. A similar application is presented in [6] which deals with the 2015 earthquake in Nepal (with magnitude 7.8 M_W on the moment magnitude scale) and the logistics for disaster relief. In the first stage, nodes are assigned to vehicles and different metaheuristics, such as a greedy

algorithm or Simulated Annealing are used to calculate paths for each vehicle in the second stage.

A slightly different, but nonetheless relevant two-stage routing problem is investigated by Zhong, Hall, and Dessouky [204], where in the first stage "core areas" of the network are assigned to drivers and in the second stage concrete tours for each driver are planned on a network with stochastic components that change over time. This is comparable to a two-step planning procedure presented above [155] where "general routes" are planned in the first step from which actual routes are derived in the second step. The optimization problem they consider is to minimize the number of drivers utilized, the total route duration and route length as well as the variation of routes over time for each driver, and a two-stage planning algorithm with Tabu Search is proposed.

Two-stage problems also occur when a vehicle routing problem is combined with another optimization problem. For example, Bortfeldt and Homberger [22] consider the combination of three-dimensional packing and vehicle routing where the number of used vehicles and the traversed distance is to be minimized while taking loading constraints for each vehicle into account. They develop a two-stage heuristic that combines routing methods with Tabu Search. Another example is the combination of production scheduling with a vehicle routing problem in [206]. In this problem, the production of goods is to be scheduled in addition to their delivery to customers, and a two-stage algorithm is proposed that incorporates a Genetic Algorithm. A combination with a picking problem from a storage hall is investigated in [126] and a literature review on similar problems with a focus on picking problems can be found in [154].

The works presented above show that there are many types of two-stage optimization problems related to vehicle routing. However, the two-stage problem considered in this thesis differs from these works in that there are exactly two "vehicles" and the two stages are formed by their respective routes where nodes v_i (or customers) need to be visited *twice* before they are considered to be "fully serviced" (and the corresponding profit $r_{J_{v_i}}$ is collected).

2.3.2 Vehicle Routing Problems with Profits

The vehicle routing with profits (VRPP) is a variant of the VRP of which many variations can be found in the literature. In this type of problem profit values are assigned to the nodes (that, e.g., correspond to customers) and the profit collected in a tour is integrated into the optimization criterion or the constraints of a problem. In this section, an overview over two particularly relevant special cases is given.

Related Works with Profit as an Optimization Criterion

In this type of problem the primary optimization criterion is the maximization of the collected profit without violating constraints. For example, El-Hajj et al. [66] propose a Particle Swarm Optimization Algorithm (PSO) for a VRPP where additional length constraints are given for segments of each vehicle's route. Another metaheuristic, a Differential Evolution algorithm for the VRPP is proposed in [187].

A VRP with *vector* profits is investigated in [99], where the minimum component of the total profit vector is to be maximized. According to Lee and Ahn [99], this problem has applications for the exploration planetary surfaces and the routing for group tours and they propose a linear programming relaxation and a column-generation technique to calculate approximate solutions. Another variation is [101] where orders consist of objects that have to be transported from given pickup points to given delivery points, with the noteworthy property that orders can be fulfilled multiple times within a time limit. For this problem, a Genetic Algorithm is proposed for maximizing the collected profit.

Vidal et al. [186] consider three different VRPs with profits (capacitated profitable tour problem, VRP with private fleet and common carrier and team orienteering) and investigate neighborhood structures of these problems. They propose Local Search algorithms as well as a Hybrid Genetic Algorithm. Two of these problems (capacitated profitable tour problem and team orienteering) are also investigated in [12, 11] where a Branch-and-Price algorithm is proposed to calculate approximate solutions. These two problems are similar in that they consider the maximization of the collected profit while fulfilling constraints related to the vehicles' capacities or their tour lengths, but the capacitated profitable tour problem additionally assumes that traversing edges in a graph incurs a cost which is subtracted from the collected profit.

An interesting variation of the VRPP is the "VRPP with consistency constraints" investigated by Stavropoulou, Repoussis, and Tarantilis [161]. The authors consider the problem where the set of nodes is partitioned into "frequent customers" (with known profits) and "non-frequent customers" (where profits are only estimated) and where routes are calculated over a planning horizon while "consistency constraints" need to be taken into account. These constraints are derived from the problem of maximizing customer satisfaction and require, for example, that the same frequent customer is always serviced by the same vehicle. The authors of [161] list various applications for this problem (and the VRPP in general) and propose an adaptive Tabu Search for this problem.

A case study for a food importer in Hong Kong is presented in [203] that deals with a multi-objective VRPP with outsourcing. In this problem, nodes are assigned to transport companies that each have their own vehicle fleet, and a Local Search algorithm is developed to minimize the tour costs and maximize the profit of the transport company that obtains the minimum profit. Another example for an application regarding blood transportation is presented in [134].

The Prize-Collecting Vehicle Routing Problem

All of the works named above deal with the problem of maximizing profit. A different, but related problem is to minimize the duration or length of a tour with the condition that the profit collected must not be lower than a given value. In the VRP literature this problem is referred to as the **Prize-Collecting VRP**. Tang and Wang [170] first introduced this problem based on the Hot Rolling Problem from the iron and steel industry and proposed an Iterated Local Search. In [27] a Prize-Collecting VRP is considered where the graph is partitioned into subsets where in each subset a minimum profit needs to be collected, and a Hybrid Genetic Algorithm as well as a Branch and Price algorithm are proposed.

Trachanatzi et al. [176] consider the Prize Collecting VRP with a focus on environmental aspects by incorporating CO_2 emissions (that depend on how a vehicle is loaded) into the target criteria and investigate the performance of several population-based metaheuristics (Firefly Algorithm, Differential Evolution, Particle Swarm Optimization) for this problem. A multi-objective variant of the Prize-Collecting VRP is considered in [202] where the length of the route is to be mini-

mized as well as the number of unvisited nodes, based on data from an iron and steel production company in China. For this problem an algorithm is presented that combines Particle Swarm Optimization as well as Tabu Search.

Another scenario for this problem that is inspired by the shipping of small packages is investigated in [162] where customers (i.e., nodes) can be assigned to subcontractors (with a non-linear cost) so that they do not need to be serviced. Stenger [162] develops a Variable Neighborhood Search for this problem and presents an extension with multiple depots in [163]. A Variable Neighborhood Search is also proposed for a Prize-Collecting VRP in [103], based on the scenario of a metal-cutting machine. The algorithm is compared with a Local Search as well as a Particle Swarm Optimization algorithm.

The problem of maximizing the profit is similar to the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ described above for the Two-Stage VRP with Profits and Buffers, and the problem where a constraint is placed on the collected profit is similar to $\mathcal{O}_{\min C_{\max},\, R \geq Q}$, although some differences should be highlighted: The works named above for the VRPP consider $m \geq 2$ vehicles with their own capacity as well as "demand" values for each node with the additional constraint that the total demand on a vehicle's tour cannot exceed its capacity. For the problem considered in this thesis, there are exactly 2 vehicles, no demand values and instead of capacities for the vehicles there is a buffer. In addition, it is not possible in the works above to visit the same node and collect its profit multiple times, whereas for the problem in this thesis a reward is only connected after a node has been visited by both vehicles in the order M_1, M_2.

2.3.3 Vehicle Routing Problems with Capacity- or Resource-based Restrictions

Various types of the VRP that incorporate limited capacities, storages or buffer-like components are actively researched. However, different terms are used in the literature to refer to these concepts. The following overview presents some common terms and research works that incorporate them into the Vehicle Routing Problem.

Vehicle Routing Problems with Capacities, Storages and Buffers

In the Capacitated Vehicle Routing Problem (CVRP), each vehicle has a limited capacity and the amount of capacity used is determined by the nodes visited by a vehicle. The capacity constraint states that the amount of capacity required for a vehicle's tour cannot exceed that vehicle's capacity. Vehicle routing problems of this type are common in the literature so that in some cases even the term "Vehicle Routing Problem" is used to refer to Capacitated VRPs, e.g., [186, 11, 206]. Some of the works mentioned in the previous sections also incorporate vehicles with limited capacities, for example, [6, 12, 101, 30, 206, 155].

The work of Borcinova [21] specifically deals with the Capacitated VRP and proposes two Mixed-Integer Linear Programming models for this problem. In [167] an Artificial Bee Colony algorithm is proposed for the Capaciated VRP. A variant of the CVRP that incorporates environmental aspects (similar to [176] where a Prize-Collecting VRP is considered) is the work by Faulin et al. [43], where existing algorithms are adapted so that they incorporate the environmental impact of each vehicle's tour.

The two-stage problem [133] mentioned above combines capacitated vehicles with storages that have limited capacity. The storage is positioned between the two stages of the problem (delivery from origin to depots and from depots to customers). The work from Kuhn, Schubert, and Holzapfel [89] is similar to this in that the delivery is done in two steps via an intermediate depot with limited capacity. However, in [89] the problem is inspired by a grocery retail scenario where many orders of small size are delivered from a central depot to retail stores with limited capacity. For this reason, the batching of orders is investigated in [89] and a General Adaptive Neighborhood Search algorithm is proposed. The two-stage problem from Ostermeier et al. [126] mentioned above also incorporates a storage with limited capacity between the two stages (picking and delivery). They propose a Variable Neighborhood Search and compare it with 2 heuristics and an exact solver.

Vehicle Routing Problems with Resource Constraints

Aside from VRPs with limited capacities, problems with *limited resources* are also actively researched in the literature. In this type of problem, resources are al-

located from a (limited or unlimited) pool for each vehicle which are required for that vehicle's tour. One of the first works that combines vehicle routing with inventory allocation is from Federgruen and Zipkin [44]. They consider a VRP where resources need to be transported to customers whose demand changes over time. Over-allocating resources incurs storage cost for the depot and the customer, whereas under-allocating resources causes shortage costs. In this work the problem of minimizing the total cost is considered and the authors propose interchange heuristics as well as decomposition procedures for a corresponding Mixed-Integer Linear Program.

In [17], a framework to describe Vehicle Routing-Allocation Problems is presented. In this type of problem, it is possible to assign nodes which are not visited by any vehicle to other nodes for a given allocation cost. This is based on the problem of providing medical care services in rural areas where not only the vehicles, but also the patients (the "customers") need to travel to other nodes (e.g., larger medical facilities) by themselves. In addition, the paper presents other applications and discusses how this problem is connected to other vehicle routing problems.

Hempsch and Irnich [69] consider a VRP with so-called "inter-tour resource constraints" where the deployed vehicles compete for *globally* limited resources. This problem is particularly relevant since the Two-Stage VRP with Profits and Buffers considered in this thesis also deals with a type of "global" buffer that both vehicles need to take into account. The authors of [69] propose a "giant-tour" model where the routes of all vehicles are connected to a long tour as well as "resource-extension functions" where additional values are added to the arcs of a graph that describe the usage of resources up to a given point on a route. In addition, they develop a local search for this type of problem.

However, some differences to the problem considered in this thesis should be noted: For the Two-Stage VRP with Profits and Buffers, both vehicles need to visit the same set of nodes and it depends on the vehicle whether the processing of a job consumes buffer space (M_1) or frees buffer space (M_2), so adding values to arcs is not sufficient to model the buffer in the problem of this thesis. In addition, whether M_1 can add a job to the buffer (i.e., whether enough free space is available) depends on the jobs released from the buffer by M_2, so the tours of M_1 and M_2 constantly interact with each other regarding the available buffer capacity. Due to this, applying the concept of giant tours and resource-extension functions to

the problems considered in this thesis would require a significant extension of the framework proposed in [69].

Another concept similar to inter-tour resource constraints is investigated in [62] under the term "resource synchronization", where resources are shared by all vehicles at the same time. The authors of that work consider a VRP with cross-docking where vehicles need to pass cross-docking points (also referred to as "transshipment points" [62]) that allow for resources to be transferred between vehicles. However, the cross-docking points have a limited capacity which is filled by the vehicles passing that point so that these locations constitute points where resource synchronization occurs. Cross-docking is extensively researched in the literature and a review can be found in [181]. For the problem in [62], the authors of that work adapt an existing matheuristic that incorporates a Large Neighborhood Search.

As can be seen from the works above, the keywords "resource" or "resource allocation" encompass several types of problems. The problem investigated in this thesis can also be considered to contain some aspects of resource allocation where M_1 allocates resources (buffer space) from a limited pool (from a buffer with limited capacity) when it finishes a job (or when it starts to process a job, depending on the buffer model used).

Finally, it should be noted that the literature overview given in this and the previous sections only focused on two-stage VRPs, VRPs with profits and VRPs with capacity- or resource-based constraints, but there exist many other types of vehicle routing problems that are actively researched. However, they strongly differ from the Two-Stage VRP with Profits and Buffers considered in this thesis so that they are not included in the literature overview, but comprehensive reviews of these and other vehicle routing problems can be found in [3, 121, 175, 185].

2.4 Preliminary Remarks on Subsequent Chapters

Although the Two-Stage VRP with Profits and Buffers and the literature presented in the previous sections belong to the category of "vehicle routing problems", it should be noted that the Two-Stage VRP with Profits and Buffers encompasses other practically relevant problems that are also actively researched in the literature not related to vehicle routing. In particular, the **Two-Machine Flow Shop Problem With Buffers** and the **Orienteering Problem** are interesting special cases for the

optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, respectively, since they have various applications to scheduling and tour planning problems in the industry, and are already *NP*-hard problems by themselves that are researched in their own areas. For this reason, it is reasonable to investigate these problems in more detail so that new insights can be gained regarding their characteristics and their relationship to the Two-Stage VRP with Profits and Buffers as a more general problem.

To this end, the analyses in the following chapters dealing with the special cases have a similar structure that is briefly described in the following. First, an overview of the state of research in the area pertaining to a special case is given by presenting research works from this area and related problems. Afterwards, theoretical properties of the optimization problem corresponding to the special case are investigated. The theoretical questions considered in these sections have already been briefly mentioned in the introductory section (see Section 1), but using the notation and the concepts introduced above they can be precisely stated as follows:

- **Problem complexity:** How hard is the problem from the perspective of computational complexity theory? Do efficiently solvable subcases exist?
- **Existence of optimal permutation schedules:** Does the set of optimal schedules for an instance of the problem always contain permutation solutions, i.e., solutions where the vehicles M_1 and M_2 visit the same nodes in the same order? Or are there cases where it is favorable that the second-stage processing of the jobs by M_2 is done in a different order than the first-stage processing by M_1?
- **The gap between permutation schedules and non-permutation schedules:** If there exist instances where the set of optimal schedules contains no permutation schedule, how large is the potential gap in solution quality between the best possible permutation schedule and an optimal non-permutation schedule?

In addition, metaheuristic algorithms for the special cases are presented that are specifically adapted to the problems. The proposed algorithms are empirically compared with other methods from the literature. All of these algorithms belong to the category of so-called "anytime algorithms", i.e., they continuously calculate solutions during their run time instead of only calculating a single solution at

the end [195]. For this reason, it is reasonable to evaluate the performance of these algorithms over the entire run time instead of only considering singular time points.

Due to the similarities between the problems considered in this thesis, it is possible to apply similar methodologies for evaluating the performance of algorithms. In particular, graphical evaluation measures are used in the following chapters that take the performance over the entire run time into account (progress curves, PC) as well as how consistent a certain solution quality is reached (empirical cumulative distribution functions, ECDF). Furthermore, the performance data of the algorithms can be aggregated over problem instances with similar properties in order to investigate how different properties in the problem instances affect the performance of the algorithms. In addition, statistical tests are used to compare the performance of algorithms at selected points in time and check whether observed differences are statistically significant.

The structure for theoretical and empirical analysis described above is used for the following two special cases in Chapter 3 and Chapter 4. Afterwards, the Two-Stage VRP with Profits and Buffers as the general problem is analyzed in Chapter 5 where similar theoretical questions are discussed and several metaheuristic methods are compared using a similar methodology.

3 The Special Case $\mathcal{O}_{\min C_{\max},\, R \geq \sum r_{J_i}}$ Without Travel Times: The Two-Machine Flow Shop Problem with Buffers

The special case considered in this chapter is derived from the Two-Stage VRP with Profits and Buffers by considering $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ (i.e., minimize the makespan while collecting a minimum score Q) with $Q = \sum r_{J_i}$ (where the sum is taken over all jobs so that it is required that *all* jobs are processed[1]) and all travel times d_{ij} set to zero, i.e., $d_{ij} = 0$ for all edges e_{ij} in the graph G. In a practical scenario, travel times being zero can be interpreted in the sense that traveling between customers takes a negligibly small time, or that all nodes and their corresponding jobs are at the same location, similar to a factory. Especially in the latter case M_1 and M_2 can be considered to be stationary machines that do not travel as opposed to the "moving machines" in the original problem and the depot node v_0 can be (informally) neglected since any node can be reached from any other node without any increase in time.

The restricted case is connected to the **Two-Machine Flow Shop with Buffers** with the additional restriction that the second machine M_2 has constant processing times $b_{J_i} = c$ for all jobs J_i. To give a brief informal description of this problem, consider a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that have to be processed on two machines M_1, M_2 (first on M_1, then on M_2). The processing is non-preemptive (i.e., it cannot be paused once it has started) and the processing of a job J_i on M_2 can only start after M_1 has finished the processing of J_i. In addition, there is a buffer with limited capacity Ω that has to be taken into account while scheduling the jobs on the machines. In particular, the buffer functions like the "intermediate buffer" or "spanning buffer" described in Section 2.2. The optimization problem is to schedule all jobs on both

[1] Recall from Section 2.2 that $r_{J_i} > 0$ holds for all jobs J_i. If this condition is not satisfied, the condition $R \geq \sum r_{J_i}$ is not equivalent to processing all available jobs.

machines such that the makespan $C_{\max}$ (i.e., the time when the last job finishes on M_2) is minimized. It can be easily seen that this problem is equivalent to the optimization problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ of the Two-Stage VRP with Profits and Buffers with $Q = \sum r_{J_i}$ and $d_{ij} = 0$ for all edges e_{ij}, so refer to the formal description in Section 2.2 for the specifics of the considered flow shop problem and the notation used in this chapter.

Due to the connections between these two problems, it is possible to visualize an instance of the Two-Machine Flow Shop Problem with Buffers using instances from the Two-Stage VRP with Profits and Buffers where the travel times on all edges are zero, see Figure 3.1 for an example that shows an instance along with an example solution for both buffer models (intermediate buffer and spanning buffer). Note how in the example for the spanning buffer model the processing of J_3 has to be delayed so that the buffer capacity is not exceeded.

In the following, an overview of literature on flow shop problems with buffers is given in Section 3.1. Theoretical properties of the problem regarding computational complexity and permutation schedules / non-permutation schedules are analyzed in Section 3.2.1. A metaheuristic algorithm to construct permutation schedules for the considered special case is presented in 3.4 and empirically evaluated in 3.5. A summary of results is given in Section 3.6. This chapter is based on published research from the author, in particular [93, 96, 57].

3.1 Review of Literature on Flow Shop Problems with Buffers

In this section, an overview of literature on flow shop problems with buffers is given. The overview is divided into multiple subsections in which works are presented that focus on a specific aspect of a flow shop problem.

3.1.1 Algorithms and Metaheuristics for Flow Shops with Buffers

A large number of metaheuristics have been proposed for flow shops with an **intermediate buffer**, i.e., a buffer that exists between each two adjacent machines M_i and M_{i+1} where each job occupies the buffer after it finishes on M_i and before it starts on machine M_{i+1}. It is known that the flow shop problem with intermediate buffers is *NP*-complete [131], so that heuristic methods are needed to efficiently calculate

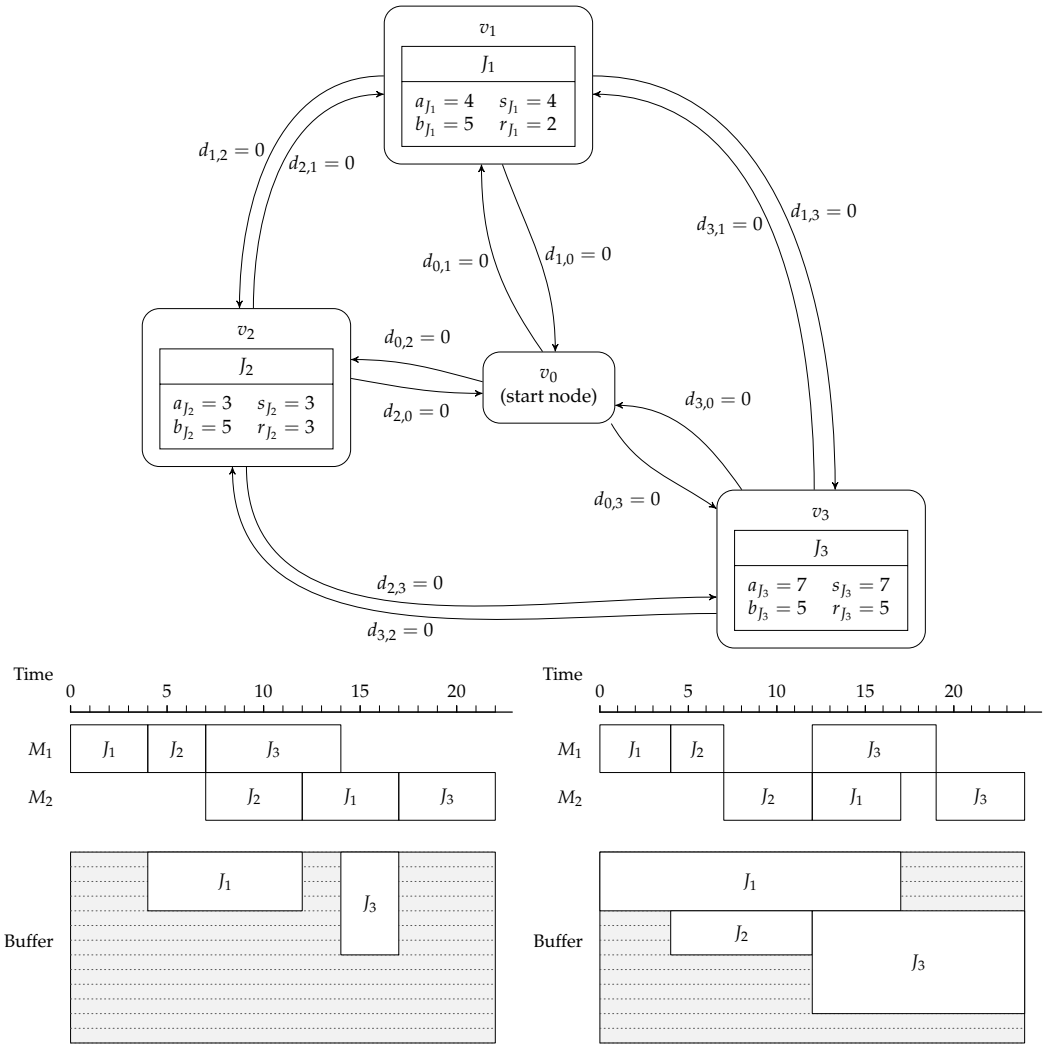
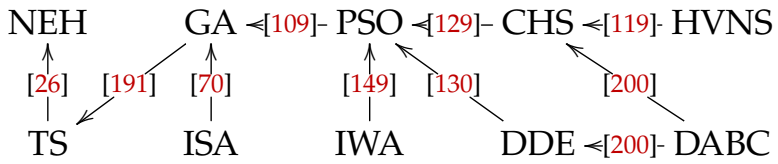


Figure 3.1: Top: Example instance for the Two-Stage VRP with Profits and Buffers with the additional property that travel times on all edges are zero. Bottom: The resulting schedules σ when M_1 and M_2 are set to process the jobs in the order $\pi^1(\sigma) = (J_1, J_2, J_3)$ and $\pi^2(\sigma) = (J_2, J_1, J_3)$ when an intermediate buffer (bottom left) or a spanning buffer (bottom right) with capacity $\Omega = 13$ is used.

Figure 3.2: Overview of works containing comparisons between algorithms [96]. The notation `A` ← `B` indicates that algorithm `A` is outperformed by algorithm `B` in the given reference. Care should be exercised when interpreting the arrows: The notation `A` ← `B` does not mean that `A` is worse than `B` for two-machine flow shops with buffer constraints. It only means that `B` obtained better results (on average) than `A` *for the considered test instances* in the respective work.



NEH: Nawaz-Enscore-Ham heuristic
GA: Genetic Algorithms
PSO: Particle Swarm Optimization
CHS: Chaotic Harmony Search
HVNS: Hybrid Variable Neighborhood Search
TS: Tabu Search
ISA: Immune System Algorithm
IWA: Invasive Weed Algorithm
DDE: Discrete Differential Evolution
DABC: Discrete Artificial Bee Colony

"good", but not necessarily optimal solutions. One of the earliest works is the work from Leisten [100] where a systematic overview for formalizing intermediate buffer flow shops problems is presented and several heuristics originating from infinite-buffer flow shops are tested with the NEH heuristic obtaining the best results. As for metaheuristics, the proposed algorithms include, e.g., a Tabu Search [104, 157], Variable Neighborhood Search [119], Genetic Algorithms [191], methods based on Differential Evolution [130], Chaotic Harmony Search [129] and Particle Swarm Optimization [109]. Other examples of biologically inspired algorithms that have been developed for this type of problem are a Discrete Artificial Bee Colony [200] and Immune System algorithms [2, 70] and [149].

In most of these studies the performance of different heuristics has been compared in experiments. Figure 3.2 summarizes the results of these comparisons [96].

Flow shops with a **spanning buffer**, (a buffer that is used by each job from its starting time on the first machine until its completion time on the last machine) are also investigated in the literature. This problem is also *NP*-complete [107] and examples for methods from the literature for this flow shop type are a Variable Neighborhood Search by Kononova and Kochetov [88], who also use Integer Linear

Programming to solve small instances, as well as a heuristic based on Lagrangian relaxation and bin packing [87]. Regarding exact methods, Lin, Hong, and Lin [107] propose a Branch-and-Bound algorithm used to calculate lower bounds and optimally solve small instances with up to 18 jobs.

A number of research works on flow shops with a spanning buffer investigate theoretical aspects of the problem, especially for the case with two machines which forms the focus of the following section.

3.1.2 Two-Machine Flow Shop Problems with Buffers

The two-machine case with spanning buffer is known to be *NP*-complete for the makespan criterion [107]. A different criterion is investigated by Gu et al. [63] who consider the total weighted completion time and show *NP*-hardness for this problem, even if one of both permutations is fixed. Min, Choi, and Park [117] prove the *NP*-hardness for the case where the processing times on both machines are the same for each job. Another variant where the spanning buffer changes its capacity over time is analyzed in [19, 18] where it is shown that this problem is *NP*-hard even when all jobs have unit processing times, but Berlińska, Kononov, and Zinder [19] present a polynomial-time approximation scheme and empirically show that smaller instances with up to 40 jobs can be optimally solved using Integer Linear Programming. A related problem with jobs whose processing can be paused is also known to be *NP*-hard [86].

The study of [205] analyses a variety of spanning buffer flow shops with two machines by differentiating whether one or both machines have constant processing times, whether the buffer usage of each job depends on its processing time, whether for each job the processing time on the first machine is always shorter, longer or equal to the time on the second machine and whether any pair of jobs can be stored in the spanning buffer. For each of the resulting 64 flow shop problems they investigate whether the makespan minimization problem is *NP*-hard or polynomially solvable.

A special case of a buffer flow shop with two machines considered in the literature is when one of the machines has constant processing times, similar to the condition $b_{J_i} = c$ for the problems considered in this thesis. Wei and Yuan [194] show that already the case without buffer restrictions is *NP*-hard for the total weighted time criterion $\sum w_i C_i$ and present an approximation algorithm with worst-case ap-

proximation ratio 2. The case with unit processing times for all jobs, which can be considered a special case of this restriction, is investigated by Ernst et al. [42]. They consider multiple identical machines (a so-called "flexible flow shop") with a spanning buffer for the case with unit processing times and where buffers are shared between adjacent machines. They show that this case is NP-hard and even NP-hard to approximate within $4/3$ of the optimal solution.

Two-machine flow shops with intermediate buffers have also been investigated, although the majority of works consider the more general m-machine case (see Section 3.1.1 above). The NP-hardness for the two-machine case is shown by Papadimitriou and Kanellakis [131] who also propose a heuristic algorithm specific to such flow shops. Dutta and Cunningham [40] present another heuristic algorithm that is based on Dynamic Programming and successive approximations. However, it should be noted that there exists a special type of flow shop that is efficiently solvable in the two-machine case. This type of flow shop is presented in the next section.

3.1.3 Blocking Flow Shops

If the buffer size is infinite, the resulting flow shop is equivalent to a flow shop problem without buffer constraints which in the case with two machines can be optimally solved in polynomial time for the makespan criterion using Johnson's algorithm [73], whereas it is NP-hard for more than two machines [54]. On the other side, if the buffer has no capacity ($\Omega = 0$) such that finished jobs on M_i immediately block M_i if M_{i+1} is busy, one obtains the **Blocking Flow Shop** which can also be solved in polynomial time for up to two machines [58] by reducing it to an efficiently solvable subcase of the Traveling Salesman Problem [139]. Blocking flow shops with three or more machines are NP-hard for the makespan criterion [52, 115]. They are NP-hard for two machines if the total flow time $\sum C_J$ (the sum over all completion times) is to be minimized [52].

Blocking Flow Shops are based on practical applications where it is not possible to temporarily store intermediate products, e.g., the production of chemical products [110] or automated manufacturing systems [83]. Various metaheuristics have been proposed for blocking flow shop problems. These include search algorithms, e.g., Tabu Search [60] and Harmony Search [15, 190] as well as biologically inspired

methods, such as Particle Swarm Optimization [105], a Cuckoo Search algorithm [188], memetic algorithms [128], differential evolution [189] or an Artificial Bee Colony algorithm [38]. Other heuristics include constructive heuristics [122, 142], a heuristic based on Dynamic Programming [16] and an Iterated Greedy algorithm by Tasgetiren et al. [172].

Table 3.1 lists algorithms that have been applied to both flow shops with buffers and blocking flow shops. Even though the buffer capacity is the only difference between these two flow shop types, Papadimitriou and Kanellakis [131] showed that introducing a buffer that can store one job can potentially reduce the makespan by the factor of up to $1/3$ when compared to an equivalent blocking flow shop (with zero buffer capacity). Note that a capacity $\Omega = 0$ is only possible for intermediate buffers, since in the case with spanning buffers no feasible schedules exist.

	Limited Buffer FS	Blocking FS
Bee algorithms	[200]	[38]
Tabu Search	[104, 124, 26]	[60]
Harmony Search	[129]	[15, 190]
Particle Swarm Opt.	[109]	[105]
Evolution-based algorithms	[39, 130, 136, 191]	[128, 189]

Table 3.1: Overview of metaheuristics that have been applied to flow shops with limited buffers ("Limited Buffer FS") as well as blocking flow shops ("Blocking FS").

A more heavily constrained version of a blocking flow shop is the "no-wait flow shop" where even the blocking of machines is not allowed so that all processing stages of a job have to be performed without interruption. This problem occurs, e.g., in the production of food products that need to be quickly packaged or processed in order to preserve freshness [110] or in the ingot production where high temperatures need to be maintained [54]. This problem is also efficiently solvable for up to two machines, but becomes *NP*-hard for three or more machines [52]. Heuristic methods specifically developed for this problem are investigated in [100, 110, 139].

Note that in blocking flow shops and no-wait flow shops the order of processed jobs is the same on all machines, i.e., only permutation schedules are allowed since it is not possible for jobs to pass other jobs due to the missing buffer. However, if the buffers with capacity $\Omega > 0$ are used, non-permutation schedules are possible

where the machines process jobs in a different order. Even though the majority of studies for flow shops with buffers discussed above (see Section 3.1.1) only consider permutation schedules, there exist research works that specifically investigate non-permutation schedules.

3.1.4 Non-Permutation Schedules

The flow shop problem where non-permutation schedules are permitted is also known to be *NP*-complete [165] for the case with the intermediate buffer (whereas the commonly cited *NP*-hardness proof of Papadimitriou and Kanellakis [131] specifically considers permutation schedules), and there exist cases where non-permutation schedules perform better than permutation schedules. For the case with infinite buffer capacity, this has been observed for the total tardiness criterion as well as criteria based on completion time, e.g., makespan or total completion time [106], although for the latter the difference also depends on the dispersion in the processing times [146, 147].

For two-machine flow shops with a spanning buffer where for each job its buffer usage is equal to its processing time on the first machine (i.e., $s_J = a_J$ for all jobs J), Min, Choi, and Park [117] show that the set of all optimal schedules contains at least one permutation schedule if all jobs have identical processing times on each machine or if the second machine dominates the first machine with respect to processing times. In [50] it is shown that for $n \leq 4$ jobs, the set of optimal schedules always contains at least one permutation schedule, whereas for instances $n > 4$ this cannot be guaranteed. Due to this, heuristics for non-permutation schedules are investigated in the literature, for example, by Leisten [100] where several heuristics are compared or [26] where a Tabu Search algorithm is proposed. A review of heuristic methods for the non-permutation flow shop (without buffer constraints) is given in [146].

The study of [145] investigates a special case with two jobs and unknown processing times with a focus on dominance properties between permutation schedules and non-permutation solutions using a graph-theoretical approach. The theoretical work of [138] analyses "critical jobs" that in optimal non-permutation schedules have to be processed at certain positions.

3.1.5 Other Extensions and Variations of Flow Shop Problems

The buffer-constrained flow shop has also been investigated with other additional constraints in the literature. A combination of multiple target criteria is investigated by [137], whereas [110] consider the combination of different buffer types for the same instance. Buffer flow shops that allow the "batch processing" of jobs are investigated in [49, 7, 135, 4].

Another extension is the use of machines that can process multiple jobs at the same time. This type is known as the "parallel flow shop", "hybrid flow shop" or "flexible flow shop" [42] problem and heuristics for this type of problem are presented in [102, 177, 158] (for flow shops with intermediate buffers) and [141, 171, 198] (for blocking flow shops). Regarding the case with two machines and spanning buffer, it has been shown that this problem is *NP*-hard [51]. A more general case consisting of multiple parallel flow shops is considered by [199] where a Differential Evolution heuristic is proposed.

A different flow shop problem which is not related buffer flow shops, but to the Two-Stage VRP with Profits and Buffers is obtained when "sequence-dependent setup times" are imposed. This means that a machine M needs to wait for at least $\delta(J_i, J_j)$ time units between the processing of two jobs J_i and J_j where the time $\delta(J_i, J_j)$ depends on the previous job J_i and the subsequent job J_j. The sequence-dependent setup times $\delta(J_i, J_j)$ are similar to the travel times d_{ij} in the Two-Stage VRP with Profits and Buffers in the sense that "setting up a machine for processing job J_j after finishing J_i" is comparable to "travelling from node v_i to v_j" where v_i and v_j are the nodes containing the jobs J_i and J_j, respectively. And even though the Two-Stage VRP with Profits and Buffers encompasses the two-machine flow shop problem with buffers (i.e., a scheduling problem is seen from the perspective of vehicle routing), the notion of sequence-dependent setup times allows some cases of the Two-Stage VRP with Profits and Buffers to be considered a "flow shop problem with sequence-dependent setup times and buffers" (i.e., a vehicle routing problem is reinterpreted as a scheduling problem). Even though it should be noted that this only describes the optimization problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ where *all* jobs need to be processed, this type of flow shop is nonetheless relevant for the problems considered in this thesis.

This problem is known to be *NP*-hard for the case with two machines [65] and mathematical heuristics for this problem are investigated in [156]. Exact methods

proposed for small instances include Dynamic Programming [35, 34] as well as Branch-and-Bound algorithms [34]. Note that these works do not consider buffer restrictions. Interestingly, metaheuristics for this problem as well as the case with buffer restrictions have not been considered in the literature so far, so that results presented in this thesis can potentially provide new insights for flow shop problems with sequence-dependent setup times due to their similarities with the Two-Stage VRP with Profits and Buffers.

3.2 Theoretical Properties

In the following sections, theoretical properties of the Two-Machine Flow Shop Problem with Buffers are investigated. As mentioned in Section 2.4, the analyses consider the computational complexity of the problem, the existence of permutation schedules in the set of optimal solutions and the gap between the best possible permutation schedule and optimal non-permutation schedules.

3.2.1 Computational Complexity

It is shown in this section that flow shop problems with buffers are *NP*-complete for both the intermediate buffer and spanning buffer case even when for the second machine all jobs have the same processing time $b_J = c$. However, there exist special cases where an optimal solution can be efficiently constructed.

To specify the problems for which the complexity is analyzed, the following parameters for Two-Machine Flow Shop Problem with Buffers, which are based on common choices in the literature, are considered:

1. type of the buffer $\mathsf{bufType} \in \{\mathit{intermediateBuffer}, \mathit{spanningBuffer}\}$: This parameter denotes the type of the buffer used in the flow shop problem (intermediate buffer or spanning buffer).

2. buffer usage for each job $\mathsf{bufUsage} \in \{s_J = 1, s_J = a_J\}$: This parameter describes whether $s_J = 1$ or $s_J = a_J$ holds for all jobs J.

3. schedule type $\mathsf{schedType} \in \{\mathit{unrestricted}, \mathit{prmu}\}$: This parameter describes whether non-permutation schedules are allowed ($\mathsf{schedType} = \mathit{unrestricted}$) or if only permutation schedules are allowed ($\mathsf{schedType} = \mathit{prmu}$).

Depending on how the three parameters bufType, bufUsage, schedType are chosen, a total of 8 flow shop types are obtained whose complexity can be analyzed. Based on the three-field notation introduced in [61], the flow shop problems are denoted

$$F2|\mathsf{schedType}, b_J = c, \mathsf{bufType}, \mathsf{bufUsage}|C_{\max}. \tag{3.1}$$

This notation refers to a two-machine flow shop problem with the restriction $b_J = c$ for all jobs J and the parameters schedType, bufType, bufUsage (which are described above), where the makespan $C_{\max}$ is to be minimized.

NP-hardness results

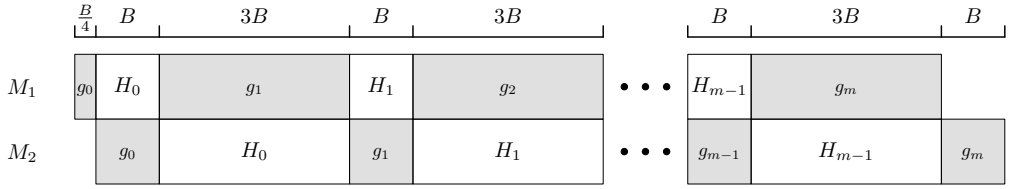
For flow shops with intermediate buffer "$s_J = 1$ for all jobs", the *NP*-completeness was already proven for the general case with non-permutation schedules [165] and for the restricted case where only permutation schedules are allowed [131], whereas for flow shops with a spanning buffer only the case with "$s_J = a_J$ for all jobs" is known to be *NP*-complete [107]. However, these cases do not consider the restriction to constant processing times on the second machine. The following theorem, which is based on published work of the author [96], states that for all configurations of the three parameters bufType, bufUsage and schedType the resulting flow shop problems are *NP*-complete, even when the second machine has constant processing times for all jobs. Thus, this theorem proves a stronger version of the aforementioned results and in addition shows the *NP*-completeness for some of the cases not yet considered in the literature.

For this theorem, a detailed proof is only given for one of the cases. The *NP*-completeness for the other considered flow shops can be shown with similar arguments, e.g., by slightly adapting the constructed instance or by generalizing some formulations, but the main idea of reducing from the same *NP*-hard problem is the same for all cases.

Theorem 3.2.1. *For all choices of* bufType, bufUsage *and* schedType, *the decision problem if, for an instance of* $F2|\mathsf{schedType}, b_J = c, \mathsf{bufType}, \mathsf{bufUsage}|C_{\max}$ *there exists a feasible schedule* σ^* *with* $C_{\max}(\sigma^*) \leq L$ *for a given integer* L *is NP-complete.*

Proof (only for the case $F2|prmu, b_J = c, intermediateBuffer, s_J = a_J|C_{\max}$*).* The problem is in *NP* since it can be checked in polynomial time if a permutation schedule

Figure 3.3: Visualization for Theorem 3.2.1



σ^* is valid and satisfies $C_{\max}(\sigma^*) \leq L$. To show the *NP*-hardness, we reduce from 3Partition: Given are positive integers $x_1, x_2, \ldots, x_{3m}$ and an integer $B > 0$ such that $B/4 < x_j < B/2$ for all j and $\sum_{j=1}^{3m} x_j = mB$. The question for 3Partition is if there exists a partition of $\{x_1, x_2, \ldots, x_{3m}\}$ into m subsets $S_1, S_2, \ldots, S_m$ such that each set S_k satisfies $|S_k| = 3$ and $\sum_{x \in S_k} x = B$. We denote the given instance from 3Partition as I_{3P}. The corresponding flow shop instance I_{F2} of the type $F2|prmu, b_J = c, intermediateBuffer, s_J = a_J|C_{\max})$ is described in the following. The set of jobs $\mathcal{J} = \{g_0\} \cup G \cup H$ consists of $n = 4m + 1$ jobs where

- g_0 is a job with $a_{g_0} = B/4$,
- G is a set of m jobs $g_1, g_2, \ldots, g_m$ with $a_{g_k} = 3B$ for $k \in \{1, 2, \ldots, m\}$, and
- H is a set of $3m$ jobs $h_1, h_2, \ldots, h_{3m}$ with $a_{h_k} = x_k$ for $k \in \{1, 2, \ldots, 3m\}$.

The additional parameters are $\Omega = 3B/4$, $c = B$, $L = 4mB + B + B/4$ and $s_J = a_J$ for all jobs. This instance can be constructed in polynomial time. Now it is shown that I_{3P} has a solution if and only if for I_{F2} there exists a permutation schedule σ^* with $C_{\max}(\sigma^*) \leq L$.

"$\Leftarrow$": Assume that I_{F2} has a permutation schedule σ^* with $C_{\max}(\sigma^*) \leq L$. It is shown in the following that σ^* satisfies several properties that lead to a structure shown in Figure 3.3. From these properties a solution for I_{3P} can be constructed. The first property results from the sum of all processing times on both machines, using $\sum_{j=1}^{3m} x_j = mB$ and comparing the values to L:

$$\sum_{j \in \mathcal{J}} a_j = \frac{B}{4} + m \cdot 3B + mB = 4mB + \frac{B}{4} = L - B$$

$$\sum_{j \in \mathcal{J}} b_j = B + mB + 3mB = 4mB + B = L - \frac{B}{4}$$

This shows that M_1 cannot have a total idle time of more than B time units and M_2 cannot be idle for more than $B/4$ time units, or else the resulting schedule exceeds the maximum makespan L. Since M_1 is always idle for at least B time units (while M_2 is processing its last job), M_1 is not allowed have any other idle times before it finishes its last job.

As for M_2, the initial idle time interval (while M_1 processes its first job) cannot be shorter than $B/4$ time units and its length is equal to $B/4$ if and only if g_0 is the first job on both M_1 and M_2 (a job $h_k \in H$ cannot be the first job due to $B/4 < x_k < B/2$ being required for the corresponding 3Partition instance). Thus, g_0 has to be the first job on both machines and no idle time is allowed on M_2 between g_0 and the last job on M_2.

The second property is that it is not possible for the buffer to store any job $g \in G$ due to $s_g = 3B > 3B/4 = \Omega$. Thus, the machine M_2 must immediately start any g-job that finishes on M_1: $C_g^1(\sigma^*) = S_g^2(\sigma^*)$. It is also necessary that $C_{g_0}^1(\sigma^*) = S_{g_0}^2(\sigma^*)$ or else the initial idle time is exceeded. If we neglect the jobs in H and only consider the jobs in G, it can be said that the order in which the jobs of G are processed is equal for both machines: If that were not the case, then there would be two jobs $g, g' \in G$ where g' is processed before g on M_1 , but after g on M_2. This would imply that g' would have to be stored in the buffer which is not possible as this would exceed the capacity Ω. Since all jobs in G are identical, it can be assumed without loss of generality that the G-jobs are processed in the order $g_1, \ldots, g_m$.

Next, consider the sets $H_k \subseteq H$ of jobs processed between g_k and g_{k+1} (for $k \in \{0, 1, 2, \ldots, m-1\}$) and the set H_m of jobs processed after g_m on both machines. Note that the condition $B/4 < x_k < B/2$ for $k \in \{1, \ldots, 3m\}$ implies that the buffer cannot store more than two jobs from H at any time. It is now shown that $|H_k| = 3$ and $\sum_{j \in H_k} a_j = B$ for $k \in \{0, 1, 2, \ldots, m-1\}$. First, consider $k = 0$. Since no additional idle times are allowed on both machines, the equations $S_{g_1}^2(\sigma^*) = C_{g_0}^1(\sigma^*) + b_{g_0} + \sum_{h \in H_0} b_h = a_{g_0} + (1 + |H_0|)B$ and $C_{g_1}^1(\sigma^*) = a_{g_0} + \sum_{h \in H_0} a_h + a_{g_1}$ as well as $C_{g_1}^1(\sigma^*) = S_{g_1}^2(\sigma^*)$ lead to $\sum_{h \in H_0} a_h = (|H_0| - 2)B$ which cannot be negative or zero, implying $|H_0| \geq 3$.

Assume $|H_0| > 3$ and denote as $h^1, h^2, h^3, h^4 \in H$ the first four jobs in H_0 processed on M_1. Machine M_2 also has to process these jobs immediately after finishing g_0 or else idle times are incurred. Using the condition $B/4 < a_h < B/2$ for all $h \in H$

leads to

$$C^1_{h^4}(\sigma^*) = C^1_{g_0}(\sigma^*) + \sum_{i=1}^{4} a_{h^i} = S^2_{g_0}(\sigma^*) + \sum_{i=1}^{4} a_{h^i} < S^2_{g_0}(\sigma^*) + \underbrace{2B}_{b_{g_0}+b_{h^1}} = S^2_{h^2}(\sigma^*).$$

This means that four jobs in H^1_0 are finished on M_1 before M_2 frees the buffer space occupied by h^2 such that at least three jobs from H_0 have to be stored in the buffer at time $t = C^1_{h^4}(\sigma^*)$. This exceeds the buffer capacity from which it follows σ^* is not a feasible solution. Thus, $|H_0| = 3$.

In order to show $\sum_{j\in H_0} a_j = B$, assume all other cases: If $\sum_{j\in H_0} a_j < B$, the three jobs in H_0 would be finished on M_1 before M_2 finishes g_0 implying that they are stored in the buffer. This is not possible since $\sum_{j\in H_0} s_j > (3/4)B$ and this would exceed the buffer capacity Ω. For the case $\sum_{j\in H_0} a_j > B$, let h^3 be the job in the set H_0 that M_2 processes last. It follows that

$$\begin{aligned} C^2_{h^3}(\sigma^*) &= S^2_{g_0}(\sigma^*) + \underbrace{B}_{b_{g_0}} + \underbrace{3B}_{\sum_{j\in H_0} b_j} = C^1_{g_0}(\sigma^*) + B + 3B \\ &< C^1_{g_0}(\sigma^*) + \sum_{j\in H_0} a_j + \underbrace{3B}_{a_{g_1}} = C^1_{g_1}(\sigma^*), \end{aligned}$$

which means that M_2 is idle (waiting for g_1) after finishing h^3. This is a contradiction since there no idle times between the jobs on M_2 are allowed.

Hence, $|H_0| = 3$ and $\sum_{j\in H_0} a_j = B$ holds. From this it follows that M_1 finishes the set H_0 at the same time as M_2 finishing g_0 and that the interval from $C^2_{g_0}(\sigma^*) = S^1_{g_1}(\sigma^*)$ to $S^2_{g_1}(\sigma^*) = C^1_{g_1}(\sigma^*)$ has a length of $3B$ time units. Thus, M_2 has to process the jobs in H_0 during this interval or else it would be idle.

It can also be concluded that the buffer is empty by the time M_1 finishes the job g_1. This allows us to apply the same arguments used in the case $k = 0$ for $k \in \{1, 2, \ldots, m-1\}$ to show that $|H_k| = 3$ as well as $\sum_{j\in H_k} a_j = B$. By iteration, one obtains m sets $H_0, H_1, H_2, \ldots, H_{m-1}$ that each contain three jobs from H. Due to $|H| = 3m$, it follows that $|H_m| = 0$. Since $\sum_{j\in H_k} a_j = B$ for $k \in \{0, 1, 2, \ldots, m-1\}$ and since the jobs in H correspond to the 3Partition numbers, it is possible to construct a solution for I_{3P}.

"⇒": Given the subsets $S_1, S_2, \ldots, S_m$ satisfying $|S_k| = 3$ and $\sum_{x \in S_k} x = B$ for $k \in \{1, 2, \ldots, m\}$, a schedule σ^* can be constructed as shown in Figure 3.3 where the H-jobs corresponding to the subsets are scheduled between the G-jobs. This leads to a schedule σ^* with $C_{\max}(\sigma^*) = L$ and a solution for I_{F2}.

□

Polynomial-Time Solvable Subcases

As stated in Theorem 3.2.1, the buffer flow shops with $b_J = c$ for all jobs J are *NP*-complete for all values of schedType, bufType and bufUsage considered here. In the following it is shown that two special subcases are solvable in polynomial time, independent of the parameters bufType, bufUsage and schedType:

- The constant c is so small that $c \leq a_J$ holds for all jobs J.
- The constant c is so large that $c \geq a_J$ holds for all jobs J.

These cases were identified in a published work from the author [96] which also forms the basis for the following analyses. The work [96] only considers permutation schedules, but a new result presented in this thesis is that optimal schedules can be found in polynomial time for these cases even if non-permutation schedules are allowed.

In order to show the property for the case $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$ (specifically with a spanning buffer where $s_J = a_J$ holds for all jobs), some lemmata are needed. The first lemma states that non-permutation schedules are equivalent to permutation schedules in terms of attainable solution quality.

Lemma 3.2.1. *Given a non-permutation schedule σ for an instance of $F2|unrestricted, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$, there exists a permutation schedule σ^P satisfying $C_{\max}(\sigma) = C_{\max}(\sigma^P)$.*

Proof. Assume that $a_J \leq c$ holds for all jobs J. For a given non-permutation schedule σ, a permutation schedule σ^P is constructed as follows. Set $\pi(\sigma^P) = \pi_2(\sigma)$ so that both machines in σ^P process the jobs in the same order as M_2 in σ. Without loss of generality, it is assumed that $\pi(\sigma^P) = (J_1, J_2, \ldots, J_n)$. Next, schedule the jobs on M_1 using the rule $S^1_{J_i}(\sigma^P) = S^2_{J_i}(\sigma) - a_{J_i}$ for all J_i. For M_2, the same starting times

as in σ are used: $S^2_{J_i}(\sigma^P) = S^2_{J_i}(\sigma)$ for all J_i. By definition, $C^1_{J_i}(\sigma^P) = S^2_{J_i}(\sigma^P)$. The resulting schedule σ^P satisfies $C_{\max}(\sigma^P) = C_{\max}(\sigma)$ and is a permutation schedule.

However, it needs to be shown that σ^P is a valid schedule, i.e., that it is possible for M_1 to start all jobs at the specified times. This is shown by induction. First, consider the first job J_1 on M_1. Note that at the starting time $S^2_{J_1}(\sigma)$ in the original schedule σ, machine M_1 must have already processed J_1 at this time, i.e.. $S^2_{J_1}(\sigma) \geq a_{J_1}$. Thus, it is possible to set $S^1_{J_1}(\sigma^P) = S^2_{J_1}(\sigma) - a_{J_i}$ in the new schedule σ^P.

For the induction step (i.e., the remaining jobs after J_1), it needs to be shown that the starting times specified for M_1 do not overlap, i.e., $S^1_{J_{i+1}}(\sigma^P) \geq C^1_{J_i}(\sigma^P)$ for $1 \leq i \leq n-1$. To show this, assume that J_i can be processed by M_1 in σ^P at the specified time $S^1_{J_i}(\sigma^P) = S^2_{J_i}(\sigma) - a_{J_i}$ (induction hypothesis). This implies $S^2_{J_i}(\sigma^P) = C^1_{J_i}(\sigma^P)$. Using this equation shows that $S^1_{J_{i+1}}(\sigma^P) = S^2_{J_{i+1}}(\sigma^P) - a_{J_{i+1}} \geq S^2_{J_i}(\sigma^P) + c - a_{J_{i+1}} = C^1_{J_i}(\sigma^P) + c - a_{J_{i+1}} \geq C^1_{J_i}(\sigma^P) + a_{J_{i+1}} - a_{J_{i+1}} = C^1_{J_i}(\sigma^P)$. In short, $S^1_{J_{i+1}}(\sigma^P) \geq C^1_{J_i}(\sigma^P)$, so M_1 is able to process J_{i+1}. Thus, the permutation Schedule σ^P is valid regarding all starting times. (Note that it is not necessary to check whether M_2 processes jobs that are not finished yet on M_1 due to the definition of σ^P).

It remains to check whether σ^P is feasible with respect to the buffer constraints. Note that at most 2 jobs J_i, J_{i+1} can enter the buffer in σ^P at any time due to the definition of σ^P. Assume that these two jobs exceed the buffer capacity, i.e., $s_{J_i} + s_{J_{i+1}} > \Omega$. Then it can be shown that in the original schedule σ the time interval $[S^2_{J_{i+1}}(\sigma), C^2_{J_i}(\sigma)]$ spans at least $a_{J_{i+1}}$ time units, which is sufficiently large so that J_i and J_{i+1} cannot both be in the buffer in σ^P. To see this, assume that the interval $[S^2_{J_{i+1}}(\sigma), C^2_{J_i}(\sigma)]$ in σ spans less than $a_{J_{i+1}}$ time units. In σ, the job J_{i+1} must have been processed on M_1 at some time before $C^2_{J_{i+1}}$. However, all possibilities for $S^1_{J_{i+1}}(\sigma)$ imply for the spanning buffer model, that J_i and J_{i+1} are both in the buffer, which contradicts the assumption that they exceed the buffer capacity. Thus, when two jobs J_i, J_{i+1} cannot be in the buffer at the same time, they cannot be in the buffer at the same time in the constructed permutation schedule σ^P. □

The same property also holds for the other case where $c \leq a_J$ for all jobs J.

Lemma 3.2.2. *Given a non-permutation schedule σ for an instance of $F2|$unrestricted, $b_J = c$, spanningBuffer, $s_J = a_J, c \leq a_J|C_{\max}$, there exists a permutation schedule σ^P satisfying $C_{\max}(\sigma) = C_{\max}(\sigma^P)$.*

Proof. Similar to the proof of 3.2.1, except that $\pi^1(\sigma)$ (instead of $\pi^2(\sigma)$) in the non-permutation schedule σ is "copied" to the permutation $\pi(\sigma^P)$ in the constructed permutation schedule σ^P. □

The two lemmata above establish that the restriction to permutation schedules does not restrict the attainable solution quality for this special case. This allows the following analyses for the case with spanning buffer and $s_J = a_J$ to be restricted to permutation schedules only. The following lemmata are needed in order to establish assumptions that can be made about schedules without losing any generality.

Lemma 3.2.3. *For each schedule σ for an instance of $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$, there exists a schedule σ' with $\pi(\sigma) = \pi(\sigma')$ and $C_{\max}(\sigma') \leq C_{\max}(\sigma)$ satisfying $C_J^1(\sigma') = S_J^2(\sigma')$ for all jobs $J \in \mathcal{J}$.*

Proof. For a given schedule σ, assume that there exists a job J_k for which $C_{J_k}^1(\sigma) < S_{J_k}^2(\sigma)$ holds. Without loss of generality let J_k be the rightmost such job so that all jobs J_ℓ after J_k satisfy $C_{J_\ell}^1(\sigma) = S_{J_\ell}^2(\sigma)$. Then, since $c \geq a_J$ for all jobs J, it follows that each job which comes after J_k in σ starts later than $S_{J_k}^2(\sigma)$ on M_1. Hence, J_k can be rescheduled on M_1 to a new schedule σ' such that $C_{J_k}^1(\sigma') = S_{J_k}^2(\sigma) = S_{J_k}^2(\sigma')$ holds without increasing the makespan of the schedule. The lemma follows by repeating these arguments. □

For the schedule σ' of Lemma 3.2.3 it holds that for each job J_k ($k \in \{1, 2, \ldots, n\}$) the predecessor job finishes at time $t \leq S_{J_k}^2(\sigma')$ and the successor job of J_k starts at time $t \geq C_{J_k}^1(\sigma') = S_{J_k}^2(\sigma')$. This implies for σ' that at most two jobs are in the buffer at any given time. A permutation schedule is defined to be *minimal* if it has the shortest makespan out of all permutation schedules with the same order of jobs. For minimal permutation schedules, the following lemma holds.

Lemma 3.2.4. *For any permutation $\pi = (J_{\pi_1}, J_{\pi_1}, \ldots, J_{\pi_n})$ of the jobs $\mathcal{J}$ in an instance of $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$ there exists a minimal permutation schedule σ where for each job J_{π_k} and its successor $J_{\pi_{k+1}}$ ($k \in \{1, 2, \ldots, n-1\}$) the permutation schedule σ satisfies either*

i) $C_{J_{\pi_k}}^2(\sigma) = S_{J_{\pi_{k+1}}}^2(\sigma)$ *and* $s_{J_{\pi_k}} + s_{J_{\pi_{k+1}}} \leq \Omega$ *or*

ii) $C_{J_{\pi_k}}^2(\sigma) = S_{J_{\pi_{k+1}}}^1(\sigma)$ *and* $s_{J_{\pi_k}} + s_{J_{\pi_{k+1}}} > \Omega$.

Proof. The lemma directly follows from Lemma 3.2.3, the property $a_{J_{\pi_{k+1}}} \leq c$ and the fact that it can be assumed that σ is a minimal permutation schedule. □

If $C^2_{J_k}(\sigma) = S^2_{J_\ell}(\sigma)$ holds for two jobs $J_k, J_\ell \in \mathcal{J}$ (where $\mathcal{J}$ is the set of all jobs) in a permutation schedule σ, we say that the processing time a_{J_ℓ} of J_ℓ on M_1 is **hidden**. Two jobs J_k, J_ℓ are **compatible** if $s_{J_k} + s_{J_\ell} \leq \Omega$. For a permutation schedule σ and its corresponding job permutation $\pi(\sigma)$, let $I(\sigma) \subset \{1, 2, \ldots, n\}$ be the set of indices i of jobs J_i which are not compatible with their predecessor, i.e., $s_{J_{\pi_{i-1}(\sigma)}} + s_{J_{\pi_i(\sigma)}} > \Omega$. The following corollary is a direct consequence of Lemma 3.2.4.

Corollary 3.2.1. *For an instance of $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$, let σ be a minimal permutation schedule and $\pi(\sigma) = (J_{\pi_1(\sigma)}, J_{\pi_2(\sigma)}, \ldots, J_{\pi_n(\sigma)})$ its corresponding permutation of jobs in $\mathcal{J}$. It holds that $C_{\max}(\sigma) = a_{J_{\pi_1(\sigma)}} + \sum_{i \in I(\sigma)} a_{J_i} + n \cdot c$.*

In the following we present Algorithm 3.1 which computes an optimal schedule for an instance of $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$. It first requires that the jobs are sorted in order of decreasing a_J which can be done in $O(n \log n)$ steps. In the case where the jobs are already sorted, the resulting algorithm runs in linear time.

In order to calculate an optimal schedule, $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$, Algorithm 3.1 considers two special cases first: i) all jobs in $\mathcal{J}$ are pairwise not compatible (lines 1 and 2) and ii) all jobs in J are pairwise compatible (lines 3 and 4). In case of (i) none of the processing times on M_1 can be hidden so that any minimal schedule σ has the makespan $C_{\max}(\sigma) = \sum_{i=1}^{n} a_J + n \cdot c$ and is thus optimal. In case (ii), it follows that $I(\sigma) = \varnothing$. In this case Corollary 3.2.1 implies that any minimal schedule σ which has J_n as its first job has the makespan $C_{\max}(\sigma) = a_n + n \cdot c$ and is optimal since $a_{J_n} \leq a_{J_i}$ for $i \in \{1, 2, \ldots, n-1\}$.

If neither (i) nor (ii) hold there exists a minimal $k \in \{2, 3, \ldots, n-1\}$ such that J_{k+1} and J_k are compatible. Since $a_{k+1} \geq a_J$ for $i > k+1$, all jobs in $\{J_{k+1}, J_{k+2} \ldots, J_n\}$ are compatible with J_k (line 6). In order for the processing time a_{J_ℓ} of a job $J_\ell \in \{J_1, \ldots, J_k\}$ to be hidden, it is necessary that a job from $\mathcal{R} := \{J_{k+1}, \ldots, J_n\}$ (line 7) is its predecessor. Clearly, in an optimal schedule the total processing time of jobs $\{J_1, \ldots, J_{k-1}\}$ on M_1 that can be hidden has to be maximal. To determine this maximum total processing time that can be hidden the following greedy approach

Algorithm 3.1 2BF-OPT for $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$

Require: $\mathcal{J} = \{J_1, \ldots, J_n\}$ with $a_J \geq a_{J_{i+1}}$ for $i \in \{1, 2, \ldots, n-1\}$, Ω

if all jobs in $\mathcal{J}$ are pairwise not compatible **then**
  return "any minimal schedule for $\mathcal{J}$ is optimal"
else if all jobs in $\mathcal{J}$ are pairwise compatible **then**
  return a minimal schedule for any order of the jobs where J_n is the first job
else
  $k \leftarrow$ smallest value in $\{2, 3, \ldots, n-1\}$ such that J_{k+1} and J_k are compatible
  $\mathcal{R} \leftarrow \{J_{k+1}, \ldots J_n\}$
  for $i = 1$ to $k-1$ **do**
    if the smallest unassigned job $J \in \mathcal{R}$ is compatible with J_i **then**
      $f(J_i) \leftarrow J$ ▷ assign J to J_i
    else
      $f(J_i) \leftarrow \varnothing$ ▷ no job is assigned to J_i
    end if
  end for
  if all jobs in $\mathcal{R}$ have been assigned **then**
    return a minimal schedule σ with $\pi(\sigma) = (J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_1), J_1)$
  else
    if there exists an $i \in \{1, 2, \ldots, k-1\}$ with $f(J_i) = \varnothing$ **then**
      $h \leftarrow$ maximum $i \in \{1, 2, \ldots, k-1\}$ with $f(J_i) = \varnothing$
    else
      $h \leftarrow 0$
    end if
    $\mathcal{R}' \leftarrow \mathcal{R} \setminus \{f(J_i) \mid i \leq h\}$
    for $i = k-1$ to $h+1$ **do**
      $J \leftarrow$ largest unmarked job in $\mathcal{R}'$ that is compatible with J_i
      $f(J_i) \leftarrow J$ and mark J ▷ $f(J_i)$ is newly defined
    end for
    $f(J_k) \leftarrow$ smallest unassigned job in $\mathcal{R}$
    return a minimal schedule σ for the job permutation
      $\pi(\sigma) = (f(J_k), R, J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_1), J_1)$ where R is any order of the jobs
      in $\mathcal{R} \setminus \{f(J_i) \mid i \in \{1, 2, \ldots, k\}\}$
  end if
end if

is taken in lines 8–14. The jobs $J_1, \ldots, J_{k-1}$ are considered in this order and a function f is introduced that assigns jobs from $\mathcal{R}$ to these jobs as predecessors as follows.

For the next job J_i, the smallest job in $\mathcal{R}$ that has so far not been assigned as predecessor to one of the jobs $J_1, \ldots, J_{i-1}$ and that is compatible with J_i is always assigned to be the predecessor of J_i (if it exists). If all jobs in $\mathcal{R}$ have been assigned (line 15), the assignment f defines the only possible assignment of jobs in $\mathcal{R}$ to jobs in $\{J_1, \ldots, J_k\}$ such that for the maximum number of jobs in $\{J_1, \ldots, J_h\}$ the processing time on M_1 can be hidden and also the total hidden processing time on M_1 of these jobs is maximized. That the greedy algorithm gives the optimal solution follows from the fact that the set of all subsets of $J_1, \ldots, J_{k-1}$ which can be hidden by assigning jobs from $\mathcal{R}$ as predecessors (where each job in $\mathcal{R}$ is assigned to at most one job in $J_1, \ldots, J_{k-1}$) is an independence system over $J_1, \ldots, J_{k-1}$ and forms a matroid together with the weight function $w(J_\ell) = a_{J_\ell}$. It follows that any minimal schedule σ for the order $(J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_1), J_1)$ has the makespan $C_{\max}(\sigma) = a_k + \sum_{i \in I(\sigma)} a_{J_i} + n \cdot c$ and is optimal (line 16). An analogous result holds for jobs in $\{J_1, \ldots, J_k\}$ if there exists an $i \in \{1, 2, \ldots, k-1\}$ with $f(J_i) = \emptyset$ (line 18) and h is the maximum such i (line 19). In this case there exists an optimal schedule σ where $(J(h), f(J_{h-1}), J_{h-1}, \ldots, f(J_1), J_1)$ is a suffix of the permutation $\pi(\sigma)$.

In line 23 of Algorithm 3.1 it holds for the remaining jobs $\mathcal{J}' = \{J_k, \ldots, J_{h+1}\}$ that their processing time on M_1 can be hidden with jobs from $\mathcal{R}' := \mathcal{R} \setminus \{f(J_i) \mid i \leq h\}$. However, in order to find an assignment for which the resulting makespan $a_\ell + (|\mathcal{R}'| + k - h) \cdot c$ (where J_ℓ is the first job) is minimal the assignment f might have to be redefined for the subset $\mathcal{J}'$. For this the following lemma is needed.

Lemma 3.2.5. *If there exists an assignment of some jobs in $\mathcal{R}'$ to the jobs $\{J_k, J_{k-1}, \ldots, J_{h+1}\}$ where each job in $\{J_k, J_{k-1}, \ldots, J_{h+1}\}$ gets assigned exactly one job $f(J_i) \in \mathcal{R}'$ such that $f(J_i)$ is compatible with J_i then there also exists such an assignment f where $f(J_k)$ is the smallest possible job. This assignment f satisfies the following property: a minimal permutation schedule for the order $f(J_k), R, J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_{h+1}), J_{h+1}$ where R is any order of the jobs $\mathcal{R}' \setminus \{f(J_i) \mid i \in \{h+1, h+2, \ldots, k\}$ is an optimal permutation schedule for the jobs in $\{J_k, \ldots, J_{h+1}\} \cup \mathcal{R}'$.*

Proof. Assume there exists no permutation schedule σ with an assignment f such that the schedule $f(J_k), R, J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_{h+1}), J_{h+1}$ is optimal, i.e., all optimal schedules use an assignment f and an order of the jobs in $\mathcal{J}'$ that is only

identical for the t rightmost elements $J_{h+t}, J_{h+t-1}, \ldots, J_{h+1}$ after which a different order $J_{i_1}, J_{i_2}, \ldots, J_{i_{k-h-t}}$ is used for the remaining jobs $J_k, J_{k-1}, \ldots, J_{h+t+1}$ in $\mathcal{J}'$:

$$\begin{aligned} &f(J_{i_1}), R,\ J_{i_1},\ f(J_{i_2}),\ J_{i_2},\ \ldots, f(J_{i_{k-h-t}}),\\ &J_{i_{k-h-t}},\ f(J_{h+t}), J_{h+t}, \ldots,\ f(J_{h+2}), J_{h+2}, f(J_{h+1}),\ J_{h+1} \end{aligned} \tag{3.2}$$

Let t be maximal such that the resulting permutation (3.2) is optimal and has the highest number of jobs identical to to the sequence $J_k, J_{k-1}, \ldots, J_{h+2}, J_{h+1}$. The makespan of the corresponding schedule is equal to $a_\ell + (|\mathcal{R}'| + k - h) \cdot c$ where ℓ is the index corresponding to the first job $f(J_{i_1})$ in the schedule.

It is now shown that that a new permutation schedule with the same makespan can be constructed where the $t+1$ rightmost elements coincide with this sequence, thereby contradicting that a maximal t exists: Since $h + t < k$, the order of the jobs in $\mathcal{J}'$ is only identical for $J_{h+t}, J_{h+t-1}, \ldots, J_{h+1}$, whereas the job J_{h+t+1} is at a different position i_j in the permutation (3.2): $J_{h+t+1} = J_{i_j}$. In addition, the first job $J_{i_{k-h-t}}$ at which the two permutations differ corresponds to a job $J_s \in \mathcal{J}'$ with $s \geq h + t + 2$, or in other words, $i_{k-h-t} \geq h + t + 2$. Since the schedule is optimal, it must hold that $f(J_{i_j})$ and $f(J_{i_{j+1}})$ are compatible with J_{i_j}. Since J_{i_j} is the largest job in $J_{i_1}, J_{i_2}, \ldots, J_{i_{k-h-t}}$, it holds that $f(J_{i_j})$ and $f(J_{i_{j+1}})$ are compatible with each job in $J_{i_1}, J_{i_2}, \ldots, J_{i_{k-h-t}}$. Therefore, the minimal schedule for the order

$$\begin{aligned} &f(J_{i_1}),\ R,\ J_{i_1},\ \ldots,\ f(J_{i_j}),\ J_{i_{j+1}},\ \ldots,\ f(J_{i_{k-h-t}}),\\ &J_{i_{k-h-t}},\ f(J_{i_{j+1}}),\ J_{h+t+1},\ f(J_{h+t}),\ J_{h+t},\ \ldots,\ f(J_{h+2}),\ J_{h+2},\ f(J_{h+1}),\ J_{h+1} \end{aligned}$$

has the same makespan $a_\ell + (|\mathcal{R}'| + k - h) \cdot c$ as σ and coincides with $t+1$ jobs from the sequence $J_k, J_{k-1}, \ldots, J_{h+2}, J_{h+1}$ contradicting the maximality of t.

Thus, there exists a permutation schedule with an assignment f such that the sequence $f(J_k),\ R,\ J_k,\ f(J_{k-1}),\ J_{k-1},\ \ldots,\ f(J_{h+1}),\ J_{h+1}$ is optimal. □

Lemma 3.2.5 shows that an optimal permutation schedule can be found with an assignment function f that assigns compatible predecessors to the jobs in $\{J_k, \ldots, J_{h+1}\}$ such that $f(J_k)$ is minimal. This is done in lines 23–26 of Algorithm 3.1 where always the largest possible job from $\mathcal{R}'$ is assigned to the next job in $\{J_{k+1}, \ldots, J_{h+1}\}$ and where the smallest remaining job is assigned to J_k afterwards. In addition, it can be seen that Algorithm 3.1 always constructs the solution σ such that the

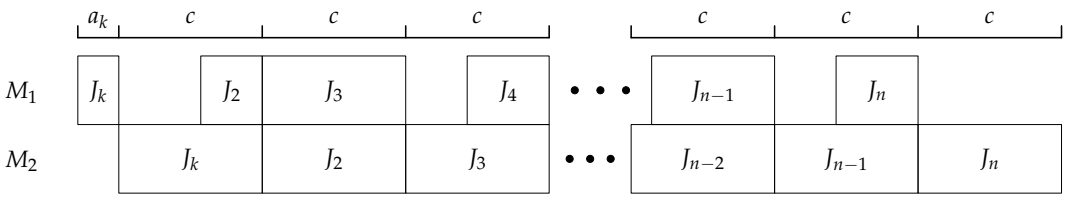


Figure 3.4: Structure of an optimal schedule for the case $F2|\mathsf{schedType}, b_J = c,$ *intermediateBuffer*$, s_J = a_J, c \geq a_J|C_{\max}$

number of neighbored pairs of compatible jobs is maximal. Thus, it is also optimal for the case where $c \leq a_{J_i}$ for all $i \in \{1, 2, \ldots, n\}$ in which case the length of the schedule is $\sum_{i=1}^{n} a_{J_i} + |I(\sigma)| \cdot c$ where $|I(\sigma)|$ is the number of neighboured pairs of incompatible jobs in $\pi(\sigma)$.

After the analysis of the cases with spanning buffer and $s_J = a_J$, the special cases (including the cases identified in [96]) that are optimally solvable in polynomial time can be stated in the following theorem.

Theorem 3.2.2. *Given an instance of* $F2|\mathsf{schedType}, b_J = c, \mathsf{bufType}, \mathsf{bufUsage}|C_{\max}$ *with* $\mathsf{schedType} \in \{$*restricted*, *prmu*$\}$, $\mathsf{bufType} \in \{$*spanningBuffer*, *intermediateBuffer*$\}$ *and* $\mathsf{bufUsage} \in \{s_J = 1, s_J = a_J\}$*, a permutation schedule that is optimal can be constructed in time* $O(n \log n)$ *if* $c \geq \max_J a_J$ *or* $c \leq \min_J a_J$.

Proof. Consider all possible values for bufType and bufUsage. It was already shown for the case with a spanning buffer, permutation schedules and $s_j = a_j$ that Algorithm 3.1 computes an optimal schedule in $O(n \log n)$ time. Lemma 3.2.1 and Lemma 3.2.2 show that this schedule is also optimal for the case $\mathsf{schedType} =$ *unrestricted*.

For the case of an intermediate buffer and $c \geq \max_J a_J$, consider the schedule σ shown in Figure 3.4 which starts with a job $J_k \in \mathcal{J}$ satisfying $a_{J_k} \leq a_{J_j}$ for all $j \in \{1, 2, \ldots, n\}$ and where each job is immediately processed on M_2 after being finished on M_1. The intermediate buffer is not occupied at any point in time. This schedule σ has the makespan $C_{\max}(\sigma) = a_k + nc$ which is a trivial lower bound for the makespan of any schedule (including non-permutation schedules). Thus, σ is an optimal permutation schedule.

Next, it is assumed that the flow shop has an intermediate buffer and satisfies $c \leq \min_i a_J$. In this case a schedule σ where all jobs are processed as early as

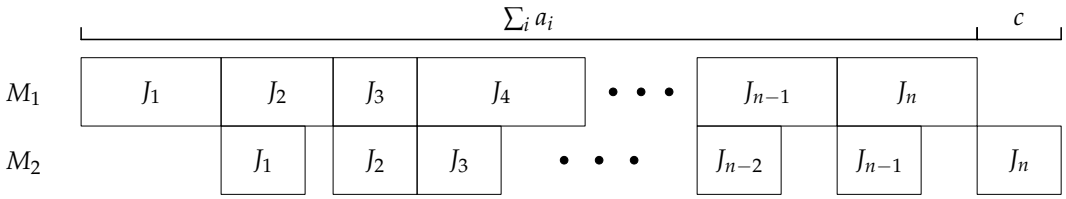


Figure 3.5: Structure of an optimal schedule for the case $F2|\mathsf{schedType}, b_J = c,$ *intermediateBuffer*$, s_J = a_J, c \leq a_J|C_{\max}$

possible must have a structure similar to the schedule shown in Figure 3.5 where the machine M_2 is always idle by the time M_1 finishes a job $J \in \mathcal{J}$. This allows M_2 to immediately start J leaving the intermediate buffer unoccupied. These schedules have a makespan of $C_{\max}(\sigma) = \sum_j a_j + c$ which is also a trivial lower bound for the makespan of any feasible schedule (including non-permutation schedules) so that all schedules of this type form optimal permutation schedules for this case.

Finally, consider the case with a spanning buffer and $s_J = 1$. Assume that $\Omega = 1$. Then, no two jobs are processed at the same time on both machines so that every permutation schedule σ (where every job starts as early as possible) is optimal with a makespan of $C_{\max}(\sigma) = \sum_i a_J + nc$. If $\Omega \geq 2$, then it is possible to construct the same schedules as in Figures 3.4 and 3.5 (depending on whether $c \geq \max_i a_J$ or $c \leq \min_i a_J$ holds) since a maximum of two jobs occupy the buffer at any time. □

3.2.2 The Existence of Optimal Permutation Schedules

This section deals with the question whether the set of optimal solutions for a flow shop with buffers contains permutation schedules. In a co-authored work [57], some properties were identified which guarantee the existence of optimal permutation schedules. Due to the excessive length of the proofs (which mainly rely on analyzing all possible cases), only the statements of the theorems are given. Detailed proofs can be found in the paper [57] and its supplement.

First, the case where $s_J = 1$ holds for all jobs J is considered. The following lemma holds for all non-permutation schedules.

Lemma 3.2.6. *Let σ be a non-permutation schedule for an instance of $F2|$unrestricted$, b_J = c,$ $\mathsf{bufType}$$, s_J = 1|C_{\max}$ with $\mathsf{bufType} \in \{$intermediateBuffer$,$ spanningBuffer$\}$. Then, there exists a permutation schedule σ^P satisfying $C_{\max}(\sigma^P) \leq C_{\max}(\sigma)$.*

This lemma can be used to directly show the following statement regarding the existence of optimal permutation schedules for that case.

Theorem 3.2.3. *For any problem of the type* $F2|unrestricted, b_J = c, \mathsf{bufType}, s_J = 1|C_{\max}$ *with* $\mathsf{bufType} \in \{intermediateBuffer, spanningBuffer\}$*, the set of optimal schedules contains at least one permutation schedule.*

When $s_J = a_J$ holds for all jobs J, the existence of optimal permutation schedules can only be guaranteed for instances with a certain number of jobs that depend on the buffer type. For two-machine flow shops with a spanning buffer *without* the restriction $b_J = c$ for all jobs J, Fung and Zinder [50] showed that optimal permutation schedules always exist for instances with up to 4 jobs, but their existence cannot be guaranteed for instances with more than 4 jobs. However, this bound changes when the restriction $b_J = c$ for all J is added.

Theorem 3.2.4. *For every problem of the type* $F2|b_J = c, spanningBuffer, s_J = a_J|C_{\max}$ *with 6 jobs or fewer, the set of optimal schedules contains at least one permutation schedule. For every* $n > 6$ $(n \in \mathbb{N})$ *there exists an instance of* $F2|b_J = c, spanningBuffer, s_J = a_J|C_{\max}$ *for which the set of optimal schedules contains no permutation schedule.*

For two-machine flow shops with an intermediate buffer, the following holds.

Theorem 3.2.5. *For every problem of the type* $F2|b_J = c, intermediateBuffer, s_J = a_J|C_{\max}$ *with 3 jobs or fewer, the set of optimal schedules contains at least one permutation schedule. For every* $n > 3$ $(n \in \mathbb{N})$ *there exists an instance of* $F2|b_J = c, intermediateBuffer, s_J = a_J|C_{\max}$ *for which the set of optimal schedules contains no permutation schedule.*

For two-machine flow shops without buffer restrictions, optimal permutation schedules always exist [73]. The results above show that introducing a buffer where $s_J = a_J$ holds for all jobs J leads to a "gap" between non-permutation schedules and permutation schedules in terms of attainable solution quality. This gap is analyzed in more detail in the following section.

3.2.3 The Gap Between Permutation Schedules an Non-Permutation Schedules

As established in the previous section, for certain flow shop problems with buffers it is possible that the set of optimal schedules contains no permutation schedule.

This raises the question how "good" the best possible permutation schedule can be when compared to an optimal schedule. In order to investigate this, some terms need to be introduced.

A permutation schedule σ^*_{perm} is a **best permutation schedule** if it has the lowest makespan out of all permutation schedules. This is to be distinguished from an **optimal schedule** σ^* that has the lowest makespan out of all schedules (including permutation and non-permutation schedules).

With these terms, the ratio $\phi := C_{\max}(\sigma^*_{perm})/C_{\max}(\sigma^*)$ that describes the ratio between the makespan of a best permutation schedule σ^*_{perm} and the makespan of an optimal schedule σ^* is analyzed in the following. In particular, instances are presented where this ratio is larger than 1. This means that the restriction to permutation schedules can lead to a gap in solution quality when compared to non-permutation schedules where even the best permutation schedule is worse than the best non-permutation schedule by a constant factor. These results are based on published work from the author [57].

In the following, the notation $C_{\max}(\sigma^*_{perm}, \mathcal{I})$ is used to denote the makespan of a best permutation schedule σ^*_{perm} and $C_{\max}(\sigma^*, \mathcal{I})$ is used to denote the makespan of an optimal schedule σ^* (out of permutation and non-permutation schedules) for a given flow shop instance $\mathcal{I}$.

Spanning Buffer

In the following, let $\mathcal{I}_k$, $k \geq 1$ be an instance of $F2|unrestricted, spanningBuffer, s_J = a_J, b_J = c|C_{\max}$ with jobs $u_i^1, u_i^2, u_i^3, r_i, g_i^1, g_i^2$ where $a_{u_i^1} = a_{u_i^2} = a_{u_i^3} = c/3$, $a_{g_i^1} = a_{g_i^2} = 2c$ and $a_{r_i} = c$ for $i = 1, \ldots, k$ and a given constant $c > 0$, and a job g_0 with $a_{g_0} = 2c$. The size of the buffer is set to $\Omega = 3c + c/3$. Jobs g_0, g_i^1 and g_i^2, $i = 1, \ldots, k$ are called jobs of type g or g-jobs. Analogously defined are jobs of type u or u-jobs and jobs of type r or r-jobs. Thus, instance $\mathcal{I}_k$ has $2k$ many u-jobs, k many r-jobs, and $2k + 1$ many g-jobs.

In order to analyze the makespan ratio between an optimal non-permutation schedule and the best permutation schedule, the following lemma specifies the length of best permutation schedules for $\mathcal{I}_k$.

Lemma 3.2.7. *For each instance $\mathcal{I}_k$, $k = 1, 2, 3, \ldots$ there exists an optimal permutation schedule σ^*_{perm} with makespan $C_{\max}(\sigma^*_{perm}) = (2k + 1) \cdot 3c + kc/3$.*

Proof. First, a lower bound on the length of an optimal permutation schedule for $\mathcal{I}_k$ is shown. Since the buffer cannot store more than one g-job exactly one of the following properties holds at any time step of a permutation schedule σ^P for $\mathcal{I}_k$:

- A: A g-job is being processed on M_1.
- B: A g-job is being processed on M_2.
- C: Neither M_1 nor M_2 are processing a g-job.

Let t_A, t_B, t_C be the number of time steps in a given permutation schedule σ^P where the aforementioned cases A, B, respectively C hold. The makespan of σ^P can be expressed as $C_{\max}(\sigma^P) = t_A + t_B + t_C$. It follows that $t_A = \sum a_g = (2k+1)2c$ and $t_B = \sum b_g = (2k+1)c$ hold and therefore $t_A + t_B = (2k+1)3c$. All schedules for $\mathcal{I}_k$ have a makespan of at least $t_A + t_B$ time units and an optimal permutation schedule needs to minimize t_C. The main idea is to calculate a lower bound $\tilde{t}_C$ for t_C from which it follows that $t_A + t_B + \tilde{t}_C$ forms a lower bound for the makespan of any permutation schedule.

Let $g_1, g_2, \ldots, g_{2k+1}$ be the order in which the jobs of type g appear in the schedule σ^P, i.e., $g_i \in \{g_0\} \cup \{g_j^1, g_j^2 \mid j = 1, \ldots, k\}$ for $i = 1, 2, \ldots, 2k+1$. Let Γ_i, $i = 1, \ldots, 2k+1$ denote the set of jobs being (fully or partially) processed on M_1 during the interval $[C_{g_i}^1(\sigma^P), C_{g_i}^2(\sigma^P)]$ in the given permutation schedule σ^P. Observe that $\Gamma_i \cap \Gamma_i = \emptyset$ holds for $i \neq j$ and $i, j \in \{1, \ldots, 2k+1\}$. This follows from the fact that the jobs in Γ_i need to be processed on M_1 before g_{i+1} is processed on M_1 and the jobs in Γ_{i+1} are processed on M_1 after g_{i+1} is processed on M_1 for $i \in \{1, \ldots 2k\}$. Observe that every job processed on M_1 after $C_{g_{2k+1}}^1(\sigma^P)$ (which includes the jobs in Γ_{2k+1}) contributes at least c time units to t_C. This holds since each of these jobs is processed on M_2 after $C_{g_{2k+1}}^2$ but no g-job is processed on M_1 during this time since g_{2k+1} is the last g-job.

Next, it is analyzed how much the jobs in the sets Γ_i for $i \in \{1, \ldots, 2k\}$ contribute to the value of t_C. Due to the limited buffer size and since σ^P is a permutation schedule, one of the following cases must hold for each of the sets Γ_i, $i = 1, \ldots, 2k$:

- case "$\varnothing$": Γ_i contains no job.
- case "u": Γ_i contains exactly one u-job.
- case "r": Γ_i contains exactly one r-job.

- case "ur": Γ_i contains exactly one u-job and one r-job (processed in the order ur or ru)
- case "uu": Γ_i contains exactly two u-jobs.
- case "uuu": Γ_i contains exactly three u-jobs.
- case "$uuuu$": Γ_i contains exactly four u-jobs.

First, consider the case "uuu". During the interval $[C^1_{g_i}(\sigma^P), C^2_{g_i}(\sigma^P)]$ none of the three u-jobs in Γ_i can leave the buffer, since M_2 has to finish g_i before processing the u-jobs. Thus, the three u-jobs need to be processed on M_2 after $C^2_{g_i}(\sigma^P)$ before the next g-job g_{i+1} can start on M_2. Since the running time of the three u-jobs on M_2 equals $3b_u = 3c$ and since processing g_{i+1} on M_1 takes only $a_{g_{i+1}} = 2c$ time units there exist at least c time units where one of the three u-jobs is processed on M_2 but no g-job is processed on M_1. Thus, Γ_i contributes at least c time units to t_C. An analogous argument shows that Γ_i for the case "$uuuu$" contributes at least $2c$ time units to t_C.

Next, consider the case "ur". Since $a_u + a_r = c + c/3$ there exist at least $c/3$ time units where jobs in Γ_i are being processed on M_1 while M_2 is not processing a g-job. Thus, Γ_i contributes $c/3$ time units to t_C in this case. However, note that the contributions to t_C caused by cases "uuu" and "$uuuu$" are due to processing times on M_2, i.e., the sum of processing times on M_2 over the jobs in Γ_i exceeds a_g, whereas the contributions to t_C caused by "ur" are due to processing times on M_1, i.e., the sum $\sum_{j\in\Gamma_i} a_j$ is larger than b_g. In the following, these two types of contributions to t_C are referred to as "t_C-contributions caused by processing times on M_1" and "t_C-contributions caused by processing times on M_2".

When determining t_C, the possibility has to be considered that a t_C-contribution of a set Γ_i caused by processing times on M_2 (cases "uuu" and "$uuuu$") can "overlap" with a t_C-contribution caused by processing times on M_1 for jobs in the set Γ_{i+1} (case "ur"). In such a case it needs to be made sure that a t_C-contribution is not counted twice for t_C. Therefore we will not count the t_C-contribution of Γ_{i+1} (case "ur") in such a case, i.e. when Γ_i is of the form "uuu" or"$uuuu$".

For the cases "$\varnothing$", "u", "r", "uu" no t_C-contribution is counted.

It should be noted that each u-job and each r-job that is not contained in any of the sets Γ_i must necessarily be processed on M_1 while no g-job is processed on

M_2. Hence, each such u-job contributes at least $c/3$ time units to t_C and each r-job contributes at least c time units to t_C. Since each such contribution to t_C is caused by processing times on M_1 it is possible that it (fully or partially) overlaps with a t_C-contribution caused by some set Γ_i which belongs to one of the cases "uuu" and "$uuuu$". Both cases are considered in the following paragraph.

First, the case "overlap between 'uuu' and jobs not contained in any Γ_i" is considered. Note that for such jobs to overlap with the t_C-contribution caused by "uuu", they have to be processed on M_1 in the interval $[C^2_{g_i}(\sigma^P), S^1_{g_{i+1}}(\sigma^P)]$, otherwise their processing on M_1 would not overlap with $[C^2_{g_i}(\sigma^P), S^2_{g_{i+1}}(\sigma^P)]$ (the time interval where the u-jobs are processed on M_2) or they would be contained in Γ_{i+1} (if they are processed in the interval $[C^1_{g_{i+1}}(\sigma^P), S^2_{g_{i+1}}(\sigma^P)]$). Thus, assume that after the three u-jobs in Γ_i a job $x \in \{r, u\}$ is processed on M_1 at some time in the interval $[C^2_{g_i}(\sigma^P), S^1_{g_{i+1}}(\sigma^P)]$. By definition, this job is not contained in any of the sets Γ_i, $j \in \{1, 2, \ldots, 2k+1\}$. Then, the contribution to t_C caused by the M_1 processing time a_x of x can overlap with the processing of u-jobs on M_2 and thus overlap the t_C-contribution caused by "uuu" by at most $a_x \leq c$ time units. However, since x is processed before g_{i+1} on M_1, it must also be processed before g_{i+1} on M_2. It follows that the interval $[C^2_{g_i}(\sigma^P), S^2_{g_{i+1}}(\sigma^P)]$ must span at least $4c$ time units, where for at least c time units none of the machines is processing a g-job and also not the x-job. Hence it can still be counted that the case "uuu" has a t_C-contribution of c time units due to processing times on M_2 (whereas for job x the t_C-contribution is still fully counted as a_x). An analogous argumentation holds if more than one job of type u and/or r is processed on M_1 during the interval $[C^2_{g_i}(\sigma^P), S^1_{g_{i+1}}(\sigma^P)]$.

Note that it is not possible that a contribution to t_C caused in this case "overlaps" with a contribution caused by the case "ur" (or vice versa) since they are both caused by processing times on M_1. Similarly, contributions to t_C caused by two sets Γ_i and Γ_{i+1} that are both of one of the cases "uuu" and "$uuuu$" (and therefore both caused by processing times on M_1) cannot overlap with each other.

Next, the problem of minimizing t_C is considered. In the following, let $v_\varnothing$, v_u, v_r, v_{uu}, v_{uuu}, and v_{uuuu} be the number of how often the aforementioned cases occur for the sets Γ_i, $i = 1, \ldots, 2k$ in the schedule σ^P. For case "ur", we introduce the variables $v_{ur/uuu}$, $v_{ur/uuuu}$, and $v_{ur/other}$ such that: i) $v_{ur/uuu}$ counts how often a set Γ_i, $i = 1, \ldots, 2k$ belongs to the case "ur" when Γ_{i-1} belongs to case "uuu", ii) $v_{ur/uuuu}$ counts how often a set Γ_i, $i = 1, \ldots, 2k$ satisfies the case "ur" when Γ_{i-1} belongs to

the case "$uuuu$", and iii) $v_{ur/other}$ counts how often a set Γ_i, $i = 1, \ldots, 2k$ is belongs to the case "ur" and none of the cases (i) or (ii) holds, i.e. either $i = 1$ or Γ_{i-1} is neither "uuu" nor "$uuuu$". In addition, let v_r^{last} and v_u^{last} be the number of r-jobs and u-jobs that are processed on M_1 after g_{2k+1}.The introduced variables need to satisfy the following equations:

- $v_r + v_{ur/uuu} + v_{ur/uuuu} + v_{ur/other} + v_r^{last} \leq k$ (there exist only k jobs of type r)
- $4v_{uuuu} + 3v_{uuu} + 2v_{uu} + v_u + v_{ur/uuu} + v_{ur/uuuu} + v_{ur/other} + v_u^{last} \leq 3k$ (there exist only $3k$ jobs of type u)
- $v_\varnothing + v_u + v_r + v_{ur/uuu} + v_{ur/uuuu} + v_{ur/other} + v_{uu} + v_{uuu} + v_{uuuu} = 2k$ (each of the sets $\Gamma_i, \ldots, \Gamma_{2k}$ is counted in exactly one of the variables)

From the first equation it can be seen that the number of "remaining" r-jobs, i.e., r-jobs that are neither contained in any set Γ_i, $i = 1, \ldots, 2k$ nor are processed after g_{2k+1} is equal to $v_r^\star := k - v_r - v_{ur} - v_{ur/uuu} - v_{ur/uuuu} - v_{ur/other} - v_r^{last}$. Similarly, from the second equations one derives that the number of of "remaining" u-jobs, i.e., u-jobs that are neither contained in any set Γ_i, $i = 1, \ldots, 2k$ nor are processed after g_{2k+1} is equal to $v_u^\star := 3k - 4v_{uuuu} - 3v_{uuu} - 2v_{uu} - v_u - v_{ur/uuu} - v_{ur/uuuu} - v_{ur/other} - v_u^{last}$.

In the following, a lower bound $\tilde{t}_C$ for t_C is calculated by solving the following minimization problem over the variables $v_\varnothing, v_u, v_r, v_{uu}, v_{uuu}, v_{uuuu}, v_{ur/uuu}, v_{ur/uuuu}$, $v_{ur/other}, v_u^{last}$, and v_r^{last} where for each case the minimum contribution to t_C is counted.

Minimize

$$\tilde{t}_C = \frac{c}{3} v_{ur/other} + c v_{uuu} + 2c v_{uuuu} + \frac{c}{3} \cdot v_u^\star + c \cdot v_r^\star + c v_u^{last} + c v_r^{last}$$

subject to the constraints

$$\begin{aligned}
v_r + v_{ur/uuu} + v_{ur/uuuu} + v_{ur/other} + v_r^{last} &\leq k \\
4v_{uuuu} + 3v_{uuu} + 2v_{uu} + v_u + v_{ur/uuu} + v_{ur/uuuu} + v_{ur/other} + v_u^{last} &\leq 3k \\
v_{ur/uuu} &\leq v_{uuu} \\
v_{ur/uuuu} &\leq v_{uuuu} \\
v_\varnothing + v_u + v_r + v_{ur/uuu} + v_{ur/uuuu} + v_{ur/other} + v_{uu} + v_{uuu} + v_{uuuu} &= 2k \\
v_\varnothing, v_u, v_r, v_{ur/uuu}, v_{ur/uuuu}, v_{ur/other}, v_{uu}, v_{uuu}, v_{uuuu}, v_r^{last}, v_u^{last} &\in \mathbb{N}
\end{aligned}$$

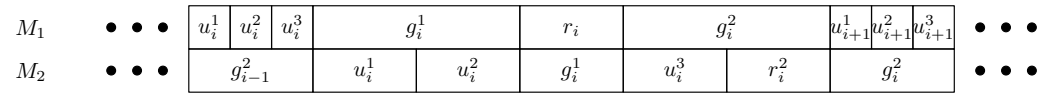


Figure 3.6: Structure of an optimal (non-permutation) schedule σ^* for an instance $\mathcal{I}_k$.

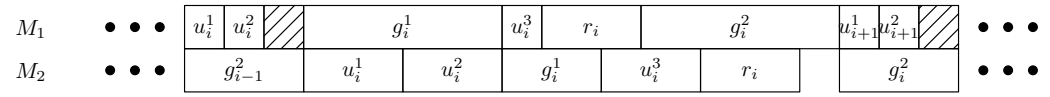


Figure 3.7: Structure of a best permutation schedule σ^*_{perm} for the instances for an instance $\mathcal{I}_k$.

The first three terms in the objective function correspond to the contribution to t_C for the cases "$ur/other$", "uuu" und "$uuuu$". The fourth term and fifth term correspond the contributions to t_C for "remaining" u-jobs and the "remaining" r-jobs, respectively. The last two terms correspond to the contribution of the u-jobs and r-jobs processed on M_1 after g_{2k+1}.

Rewriting the objective function leads to an integer linear programming problem. A straightforward application of the simplex algorithm on the relaxed problem (without integrality constraints) shows that a minimum is $\tilde{t}^*_C = kc/3$. One possible solution that leads to this value is $v_{ur/other} = k, v_{uu} = k$ where all other variables are equal to zero (Observe, that the optimal schedule which is given in the next paragraph corresponds to these values). This shows that the smallest attainable value for $\tilde{t}_C$ is the same for the linear program with integrality constraints. Thus, $t_A + t_B + \tilde{t}_C = (2k+1) \cdot 3c + kc/3$ is a lower bound for the makespan of an (optimal) permutation schedule.

To show that the lower bound is sharp an example for an optimal permutation schedule is given in the following. Let σ^*_{perm} be a schedule that starts with g_0, followed by a blockwise arrangement of jobs as shown in Figure 3.7 with length $6c + c/3$ per block. This leads to the makespan $C_{\max}(\sigma^*_{perm}) = 2c + k \cdot (6c + c/3) + c$. □

Now we can show the following theorem about the makespan ratio between a best permutation schedule and an optimal schedule for flow shop instances $\mathcal{I}_k$.

Theorem 3.2.6. *For flow shop instances $\mathcal{I}_k$ (as defined above) of the type F2|unrestricted, spanningBuffer,$s_J = a_J, b_J = c$|$C_{\max}$ with $k \geq 100$ the following holds:*

$$\frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} \geq 1 + \frac{1}{20}.$$

Proof. The instances $\mathcal{I}_k$ are chosen as described above. It was shown in Lemma 3.2.7 that a best permutation schedule σ^*_{perm} for $\mathcal{I}_k$ exists with $C_{\max}(\sigma^*_{perm}) = k \cdot (6c + c/3) + 3c$.

In the following we define an optimal (non-permutation) schedule σ^*. Let σ^* start with a g-job (which, without loss of generality, can be assumed to be g_0), followed with a job structure as shown in Figure 3.6. In this structure, each block has a length of $6c$ time units which is equal to the lower bound since each block contains 6 jobs. Schedule σ^* has makespan $C_{\max}(\sigma^*) = k \cdot 6c + 3c$ which is equal to the trivial lower bound $\sum_J a_J + c$.

To show the theorem consider $\mathcal{I}_k$ and assume $k \geq 100$ holds. Then, $C_{\max}(\sigma^*_{perm}) = (2k+1)3c + kc/3 \geq (19/3)kc$ and $C_{\max}(\sigma^*) = 6kc + 3c \leq (6c + 3c/k)k \leq ((6c + (3/100)c)k = (603/100)ck$. Hence,

$$\frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} \geq \frac{19/3}{603/100} \geq 1 + \frac{1}{20}.$$

□

For increasing k the ratio between the best permutation schedule and the optimal schedules increases monotonically. For $k \to \infty$, it is straightforward to show the following.

Corollary 3.2.2. *For the flow shop instances $\mathcal{I}_k$ as defined above of the type F2|unrestricted, spanningBuffer,$s_J = a_J, b_J = c$|$C_{\max}$, the following holds for $k \to \infty$:*

$$\lim_{k\to\infty} \frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} = \lim_{k\to\infty} \frac{k \cdot (6c + c/3) + 3c}{k \cdot 6c + 3c} = \frac{19}{18}$$

Intermediate Buffer

A similar example exists for flow shops with an intermediate buffer. Let $\mathcal{I}_k$, $k = 1, 2, 3, \ldots$ be an instance of type $F2|intermediateBuffer, s_J = a_J, b_J = c|C_{\max}$ with $3(k+1)$ with jobs h_i, u_i, and g_i where $a_{u_i} = p$, $a_{h_i} = c - p$, $a_{g_i} = 2c$ for $i = 1, \ldots, k+1$ and where for a given buffer size Ω the values p and c are chosen such that $p \leq \Omega < c - p$. Jobs h_i, (u_i, g_i, respectively) $i = 1, \ldots, k+1$ are called jobs h-jobs (u-jobs, g-jobs, respectively).

Due to the choice of the processing times on M_1 and the buffer size Ω, only a small number of different cases can occur regarding which jobs are scheduled on M_2 when M_1 starts to process a job in a permutation schedule σ^P. These cases and when they occur can be visualized using a state transition diagram shown in Figure 3.8 where the cases correspond to "states" and a "state transition" (arrow) corresponds to the next job scheduled in σ^P. These state transitions can be verified in a straightforward manner by determining the resulting schedules for each arrow (i.e., for each job that is processed next). Note that it is not possible that the idle times induced by state transitions overlap with each other.

In this diagram, each permutation schedule σ^P corresponds to a connected path that starts with the top-most arrow and consists of $k+1$ transitions for each type of jobs u, g, and h. Also, for each such path a corresponding feasible permutation schedule exists where the sequence of jobs is the same as the sequence of jobs corresponding to the arrows of the path and the total amount of idle times on one machine is the sum over the idle times of the corresponding arrows, however, for M_1 an idle time with length c has to be added that always occurs at the end of the schedule while M_2 is processing the last job.

In the following lemma a lower bound on idle times in a feasible permutation schedule for $\mathcal{I}_k$ is shown.

Lemma 3.2.8. *For each instance $\mathcal{I}_k$, $k = 1, 2, \ldots$ the total idle time on M_1 in a feasible permutation schedule is at least $k \cdot p + c$.*

Proof. Since the number of h-jobs in $\mathcal{I}_k$ is $k+1$, the path corresponding to a feasible permutation schedule σ^P must contain at least $k+1$ transitions with h-jobs. The h-transition with the smallest idle time in M_1 is $A \xrightarrow{h} A$ (excluding h being the first job in the permutation schedule σ^P). If σ^P starts with h and all h-jobs are used for this transition, at least kp time units of idle time on M_1 are incurred. (If the schedule

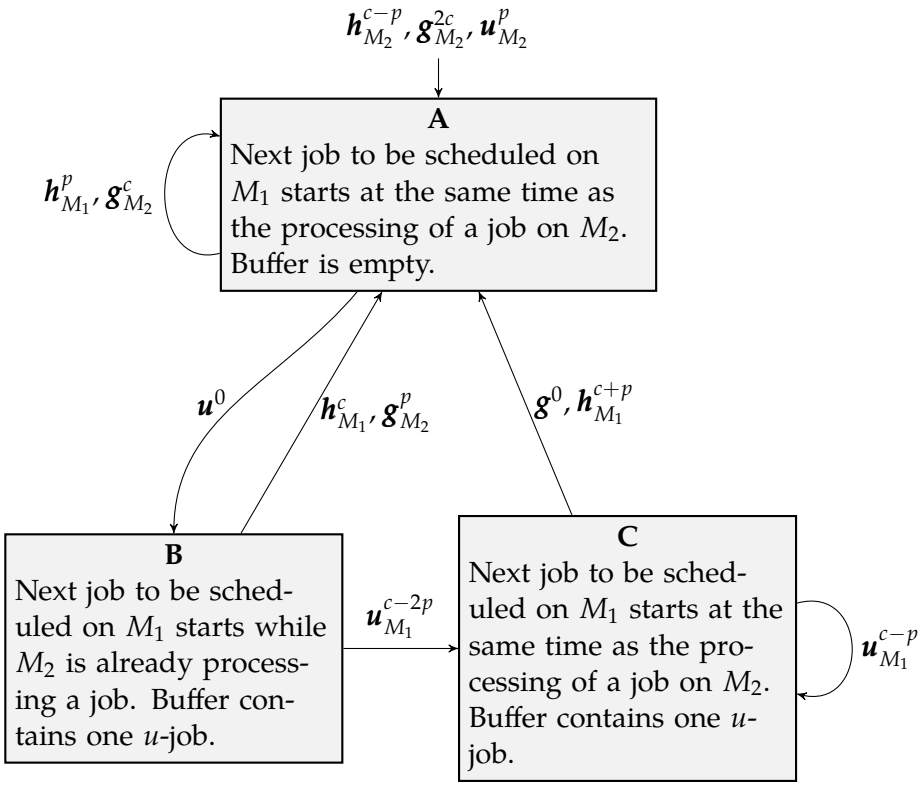


Figure 3.8: Diagram to visualize the different cases ("states") when scheduling a job j as the next job in a feasible permutation schedule σ^P and the incurred idle times on M_1 and M_2 for the instances $\mathcal{I}_k$. The symbols A, B, and C are used to abbreviate the three possible states. The first job in a permutation schedule σ^P corresponds to the top-most arrow. The notation $j_{M_i}^x$ for $j \in \{h, u, g\}$ on an arrow indicates that scheduling j in this state as the next job will incur x units of idle time on M_i, whereas j^0 indicates that no idle times are incurred on any of the machines. A feasible permutation schedule σ^P for $\mathcal{I}_k$ corresponds to a connected path through this diagram starting at the top that contains exactly $k+1$ transitions using h-jobs, $k+1$ transitions with g-jobs and $k+1$ transitions with u-jobs and the total idle time in σ^P can be calculated by summing the idle times on the traversed edges.

does not start with h, the total idle time on M_1 caused by h-jobs is at least $p(k+1)$.) Thus, the total idle time on M_1 is bounded from below by $kp + c$. □

The next lemma shows that the same lower as in Lemma 3.2.8 holds for idle times on M_2.

Lemma 3.2.9. *For each instance $\mathcal{I}_k$, $k = 1, 2, \ldots$ the total idle time on M_2 in a feasible permutation schedule is at least $k \cdot p + c$.*

Proof. Referring to Figure 3.8 above let XyZ be the number of times the edge $X \xrightarrow{y} Z$ is traversed on the path that corresponds to a permutation schedule σ^P.

In order to show the lower bound, the following property is shown first: Each occurrence of a transition $C \xrightarrow{g} A$ leads to c corresponding idle time steps on M_2. In order to show this, the argument is as follows (using Figure 3.8): If $CgA > 0$ in a permutation schedule σ^P, there are at least CgA transitions of type $A \xrightarrow{u} B$ that are not followed by $B \xrightarrow{g} A$. In addition, at least CgA transitions of type $B \xrightarrow{u} C$ must have occurred. Thus, since there are only $k+1$ jobs of type u, the following holds for the number of times the transition $A \xrightarrow{u} B$ occurs in σ^P: $AuB \leq k + 1 - CgA$. Furthermore, $BgA + CgA \leq AuB$ holds (each $B \xrightarrow{g} A$ and $C \xrightarrow{g} A$ transition requires that a $A \xrightarrow{u} B$ transition has occurred previously). It follows that $BgA + CgA \leq k + 1 - CgA$. Since there are $k+1$ jobs of type g and $0gA + AgA + BgA + CgA = k + 1$ (where $0gA$ is defined to be 1 if σ^P starts with a g-job and otherwise it is 0), the schedule σ^P must contain g-transitions that are neither $B \xrightarrow{g} A$ nor $C \xrightarrow{g} A$.

Using the above inequalities, one obtains $0gA + AgA = k + 1 - (BgA + CgA) \geq k + 1 - (k + 1 - CgA) = CgA$, which implies that σ^P contains at least $CgA \cdot c$ units of idle time on M_2 due to transitions $A \xrightarrow{g} A$ and $0 \xrightarrow{g} A$ (the latter occurs when σ^P starts with a g-job). Thus, every transition $C \xrightarrow{g} A$ leads to an idle time of at least c time units on M_2.

Now, the lower bound for idle times on M_2 is analyzed by considering all possible starting jobs. First, assume that σ^P starts with an u-job. Since the number of u-jobs in $\mathcal{I}_k$ is $k+1$, having σ^P start with a u-job implies $AuB < k + 1$. Since $BgA + CgA \leq AuB$ (as shown in the former paragraph), $BgA + CgA < k + 1$ holds. Using $0gA + AgA + BgA + CgA = k + 1$ (where $0gA = 0$ holds) leads to $AgA = k + 1 - (BgA + CgA) > 0$. This implies that there must be at least one

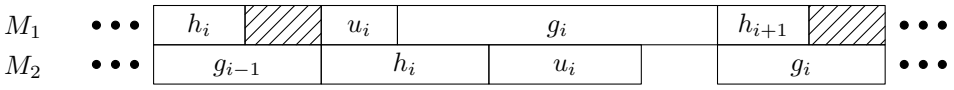


Figure 3.9: Structure of a best permutation schedule σ^*_{perm} for the instances $\mathcal{I}_k$.

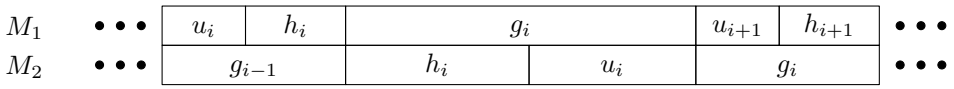


Figure 3.10: Structure of an optimal non-permutation schedule σ^* for the instances $\mathcal{I}_k$.

transition $A \xrightarrow{g} A$ in the schedule with corresponding idle time of c time steps on M_2. In the best case, the feasible permutation schedule σ^P uses the $k+1$ jobs of type g in $\mathcal{I}_k$ as follows: One transition $A \xrightarrow{g} A$ (inducing an idle time of c on M_2) and k transitions of type $B \xrightarrow{g} A$, inducing kp units of idle time on M_2 (recall that it was shown above that $C \xrightarrow{g} A$ induces more idle times than that, so it is not further considered here). By summing the resulting M_2 idle times, one obtains $p + c + k \cdot p$ (i.e., a u-transition at the beginning, one $A \xrightarrow{g} A$-transition and k transitions $B \xrightarrow{g} A$) as a lower bound for idle times on M_2 when σ^P starts with a u-job.

Next, assume that σ^P starts with a g-job (inducing $2c$ units of idle time on M_2), a simple lower bound can be obtained by assuming that the remaining k transitions with g-jobs are of the type $B \xrightarrow{g} A$. In this case summing the idle times induced by g-jobs on M_2 leads to the lower bound $2c + kp$ when σ^P starts with g.

Finally, assume that σ^P starts with an h-job and all $k+1$ transitions with g-jobs are $B \xrightarrow{g} A$, the lower bound $kp + c$ is obtained. Taking the minimum over the lower bounds for the three possible starting jobs leads to the lower bound $kp + c$. □

Consider the feasible permutation schedule σ^*_{perm} that processes the jobs in the order $h_1, u_1, g_1, h_2, u_2, g_2, \ldots, h_{k+1}, u_{k+1}, g_{k+1}$ leading to a block structure as shown in Figure 3.9 so that each block (except the first block) takes $3c + p$ time units which is a difference of length p when compared to σ^* (note the dashed lines in Figure 3.9 where M_1 is blocked). This feasible permutation schedule has idle times $kp + c$ on M_1 and M_2. Applying Lemma 3.2.8 and Lemma 3.2.9 shows that σ^*_{perm} has the minimum idle time that can be attained with feasible permutation schedules, i.e., it is a best permutation schedule with makespan $3c(k+1) + pk + c$.

Next, the following property can be shown about the makespan ratio between a best permutation schedule and an optimal (non-permutation) schedule for flow shop instances $\mathcal{I}_k$.

Theorem 3.2.7. *For flow shop instances $\mathcal{I}_k$ (as defined above) of the type F2|unrestricted, intermediateBuffer, $s_J = a_J, b_J = c|C_{\max}$ with $k \geq 25$, $p = \Omega \geq 1$, $c = 2p + 1$ the following holds:*

$$\frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} \geq 1 + \frac{1}{20}$$

Proof. The instances $\mathcal{I}_k$ are chosen as described above. It was shown above that a best permutation schedule σ^*_{perm} for $\mathcal{I}_k$ exists with makespan $C_{\max}(\sigma^*_{perm}) = 3c(k + 1) + pk + c$.

In the following we define an optimal (non-permutation) schedule σ^*. In σ^* the jobs are arranged in "blocks", see Figure 3.10. Since each block of jobs (u_i, h_i, g_i) contains three jobs, the minimal length of such a block is $3c$, from which it follows that the arrangement shown in Figure 3.10 corresponds to an optimal (non-permutation) schedule σ^* since each block has length $3c$ and the makespan is $3c(k + 1) + c$.

To show the ratio, consider $\mathcal{I}_k$ and assume $k \geq 25$, $p = \Omega \geq 1$ and $c = 2p + 1$ holds. Then $p \geq (1/3)c$ holds. Clearly, $C_{\max}(\sigma^*_{perm}) = 3c(k + 1) + pk + c \geq 3ck + pk + 4c \geq (10/3)ck$ and $C_{\max}(\sigma^*) = 3ck + 4c \leq 3ck + (4/25)ck \leq (79/25)ck$. Hence,

$$\frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} \geq \frac{10/3}{79/25} \geq 1 + \frac{1}{20}.$$

□

Similar to the example with spanning buffers, the ratio increases monotonically. By a straightforward calculation and using $p < 2c$ (which follows from the definition of the instances $\mathcal{I}_k$), the following can be shown for $k \to \infty$.

Corollary 3.2.3. *For the flow shop instances $\mathcal{I}_k$ as defined above of the type F2|unrestricted, intermediateBuffer,$s_J = a_J, b_J = c|C_{\max}$, the following upper bound for the ratio holds for $k \to \infty$:*

$$\lim_{k\to\infty} \frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} = \lim_{k\to\infty} \frac{3c(k + 1) + pk + c}{3c(k + 1) + c} = 1 + \frac{p}{3c} < \frac{7}{6}$$

The ratios presented in the two examples above for buffer flow shops show that in general it cannot be assumed that the ratio between best permutation schedules and optimal schedules is smaller than these values, i.e., they form a type of "lower bound" for the ratios that can be potentially reached. An upper bound of 2 for this ratio is established in the following which shows that the ratio cannot exceed this value. This is done by presenting an efficiently computable 2-approximation algorithm.

Theorem 3.2.8. *For any flow shop instance of the type* $F2|$*unrestricted,* $\mathsf{bufType}, s_J = a_J, b_J = c|C_{\max}$ *with* $\mathsf{bufType} \in \{$*intermediateBuffer, spanningBuffer*$\}$*, the makespan ratio between a best permutation schedule* σ^*_{perm} *and an optimal permutation schedule* σ^* *is bounded from above by*

$$\frac{C_{\max}(\sigma^*_{perm})}{C_{\max}(\sigma^*)} \leq 2.$$

Proof. Since in any feasible schedule (including an optimal schedule σ^*) all jobs need to be processed on both M_1 and M_2, a trivial lower bound is $C_{\max}(\sigma^*) \geq \max\{\sum_J a_J, \sum b_J\}$. The trivial permutation schedule $\hat{\sigma}$ that processes all jobs subsequently without overlapping satisfies $C_{\max}(\hat{\sigma}) = \sum_J a_J + \sum b_J$. From this, it follows that $C_{\max}(\sigma^*_{perm}) \leq C_{\max}(\hat{\sigma}) = \sum_J a_J + \sum b_J \leq 2 \cdot \max\{\sum a_J, \sum b_J\} \leq 2 \cdot C_{\max}(\sigma^*)$.

It remains to check that the permutation schedule $\hat{\sigma}$ does not exceed the buffer capacity at any time. In the case of an intermediate buffer, the buffer is not used at any time. In the case with a spanning buffer, not more than $\max_J s_J$ units of buffer capacity are used. Since the buffer capacity Ω cannot be less than $\max_J s_J$ in flow shops with a spanning buffer (otherwise no feasible schedules exist), $\hat{\sigma}$ is also feasible if spanning buffers are used. Thus, $\hat{\sigma}$ is a feasible permutation schedule that cannot be worse than an optimal schedule σ^* by a factor larger than 2.

□

After analyzing theoretical properties of flow shop problems with buffers, a heuristic is presented in the following section for Two-Machine Flow Shops with Buffers and constant processing times on the second machine.

3.3 A Modification of the NEH Heuristic

The NEH algorithm (named after the authors Nawaz, Emory Enscore, and Ham [122]) is a commonly used and well-known heuristic for makespan minimization in permutation flow shop problems. The algorithm starts with an empty permutation that is iteratively filled by inserting jobs (in decreasing order of the sums of their processing times) into the positions which lead to the smallest increases in makespan. It can be shown that this heuristic also calculates an optimal solution for all special cases considered in Theorem 3.2.2 since it also performs greedy insertions maximizing the number of hidden jobs as well as the sum of their processing times.

Several variants of the NEH heuristic have been studied in the literature. The majority of them deal with ties [46] or use different orders of job insertions, e.g., based on the moments of processing times [111] or using genetic programming [184]. A disadvantage of the NEH heuristic is its relatively large run time since it checks $\Theta(n^2)$ insertion points. In particular, when the solution of the NEH heuristic is further used by improvement heuristics, e.g., as the starting solution for an Iterated Local Search heuristic, it is desirable to have a faster heuristic for the computation of a starting solution.

In this section, a modified NEH heuristic (mNEH) is introduced that has a better time complexity and is suited for the considered two-machine flow shops with buffers where all processing times on the second machine M_2 are equal. In this case, the jobs differ only by their processing time on M_1 making it more likely that many jobs are similar or even identical. This property is used in the mNEH heuristic to reduce the number of positions to check for the insertions of the jobs, and thus the total number of evaluated schedules. The main idea is to split the n jobs into $G(n)$ groups of similar jobs and to maintain for each group a list of $L(n)$ "good" positions as candidates for insertion operations. This is based on the assumption that insertions of similar jobs at the same positions lead to similar changes in the resulting makespan.

A pseudocode of the modified NEH heuristic is shown in Algorithm 3.2. The number of evaluations to be performed can then be adjusted by appropriate choice of $L(n)$ and $G(n)$. In the following experiments, $L(n) = 2\sqrt{n}$ positions are memorized, i.e., potentially $\sqrt{n}$ before and $\sqrt{n}$ after the newly inserted job for each of the $G(n) = \sqrt{n}$ groups of equal size (an exception is the last group as described later).

Algorithm 3.2 Modified NEH heuristic (mNEH)

Input: number of groups $G(n)$, number of positions $L(n)$ to memorize

1: $\pi^{sort} \leftarrow$ sorted sequence of jobs by descending a_i
2: $\pi \leftarrow$ empty permutation
3: $S_1, S_2, \ldots, S_{G(n)} \leftarrow$ partition of π^{sort} into $G(n)$ groups of equal size
4: **for** $S \in \{S_1, S_2, \ldots, S_{G(n)-1}\}$ **do**
5: $(j_1, j_2, \ldots, j_k) \leftarrow$ jobs in the current group S
6: test insertion of j_1 in all possible positions of π and memorize the best $L(n)$ insertions
7: insert j_1 at the best position in π
8: update list of memorized $L(n)$ positions
9: **for** $\ell \in \{2, 3, \ldots, k\}$ **do**
10: test insertion of j_ℓ into $\hat{\pi}$ at the memorized $L(n)$ positions
11: test insertion of j_ℓ into neighbor positions of $j_{\ell-1}$
12: insert j_ℓ at the best tested position in π
13: update list of memorized $L(n)$ positions (if necessary)
14: **end for**
15: **end for**
16: insert the jobs in $S_{G(n)}$ into π as in the standard NEH heuristic
17: **return** π

With this parameter choice, the resulting algorithm performs $O(n\sqrt{n})$ evaluations, but note that the incomplete permutation π contains less than $L(n)$ jobs during the scheduling of the first two groups S_1 and S_2. For the jobs in these two groups, the mNEH heuristic tests all possible insertion positions in the same way as in the standard NEH heuristic. For the last group $S_{G(n)}$, all possible positions are checked as this group contains the shortest jobs allowing for a finer optimization of the partial permutation π. Regarding the time complexity (including the time to evaluate a permutation as well as the time to build and update the list) it can be shown that the mNEH heuristic with the used parameter values is faster than the standard NEH algorithm by a factor of $O(\sqrt{n})$.

The modified NEH heuristic is used in a metaheuristic algorithm for permutation flow shops with buffers that was developed in a published work from the author [96]. This metaheuristic is described in the following section.

3.4 An Iterated Local Search for the Two-Machine Flow Shop Problem with Buffers

The algorithm proposed for $F2|prmu, \text{bufType}, \text{bufUsage}, b_J = c|C_{\max}$ (where bufType can be an intermediate buffer or a spanning buffer and where bufUsage can be $s_J = 1$ or $s_J = a_J$ for all jobs J) is an Iterated Local Search (denoted 2BF-ILS in the following) that uses the following local search operations:

- inserting a job on position i into another position j (*insert*)
- inserting a pair of adjacent jobs at the positions $(i, i+1)$ into the positions $(j, j+1)$ (*pairInsert*)
- swapping two jobs at the positions i and j (*swap*)

The naming of these operations is based on the work of Zhang and Gu [200] and Moslehi and Khorasanian [119]. The actual selection of these operations and the order in which they are used in the experiments is later determined by the algorithm configurator `irace` in Section 3.5.3. For this reason, the following description is based on the generalized case where a sequence $op_1, op_2, \ldots, op_\ell$ is given with $op_i \in \{insert, pairInsert, swap\}$ for $i \in \{1, 2, \ldots, \ell\}$. Each of the considered operations $op \in \{insert, pairInsert, swap\}$ takes two parameters i and j and the result of op applied on π with these parameters is denoted $op(\pi, i, j)$. The resulting neighborhood of permutations around π is denoted $N^{op}(\pi) = \{\pi' \mid \exists i, j : \ op(\pi, i, j) = \pi'\}$, i.e., the set of all permutations π' that can be obtained by a single application of op on π. The size of the neighborhoods for the considered operations is quadratic with respect to the number of jobs n since the number of possible values for the parameters i and j linearly increases with n. One way to reduce the size is to fix a parameter i:

$$N_i^{op}(\pi) = \{\pi' \mid \exists j : \ op(\pi, i, j) = \pi'\}$$

This lowers the number of permutations that need to be checked, leading to potentially good solutions being missed. However, the linear size of the resulting neighborhood allows for more local search operations to be performed in the same amount of time.

The main steps of the proposed method are outlined in Algorithm 3.3: 2BF-ILS starts with the solution obtained from the mNEH heuristic proposed in Section 3.3

Algorithm 3.3 2BF-ILS

Input: initial perturbation strength ps_{init}, sequence of operations $(op_1, op_2, \ldots, op_\ell)$

$\pi^0 \leftarrow$ permutation generated by mNEH (Algorithm 3.2)
$ps \leftarrow ps_{init}$
while termination criterion not satisfied **do**
    $\pi^{cur} \leftarrow$ best known solution
    $\pi^{rand} \leftarrow$ random job permutation
    if best solution did not improve in previous iteration **then**
        $\pi^{cur} \leftarrow perturb(\pi^{cur}, ps)$
    end if
    for $s \in \{1, 2, \ldots, \ell\}$ **do**
        $op \leftarrow op_s$
        repeat ▷ local search
            for $k \in \{1, 2, \ldots, n\}$ **do**
                $i \leftarrow k$th element in π^{rand}
                $\hat{\pi} \leftarrow$ best permutation in the neighborhood $N_i^{op}(\pi^{cur})$
                if $C_{\max}(\hat{\pi}) \leq C_{\max}(\pi^{cur})$ **then**
                    $\pi^{cur} \leftarrow \hat{\pi}$
                end if
            end for
        until π^{cur} does not improve
    end for
    if best known solution did not improve in current iteration **then**
        $ps \leftarrow ps + \varepsilon$
    else
        $ps \leftarrow ps_{init}$
    end if
end while
return best known solution

(with $G(n) = \sqrt{n}$ and $L(n) = 2\sqrt{n}$). Afterwards, the algorithm iterates through a sequence of operations $(op_1, op_2, \ldots, op_\ell)$ and repeatedly performs local search steps with changing neighborhoods. The choice of the fixed parameter i in $N_i^{op}(\pi)$ is based on a random permutation π^{rand} which is calculated beforehand. The search steps with the current operation op are repeated until the permutation π^{cur} obtained so far cannot be further improved. Then, the local search procedure is restarted with the next given operation. The changing neighborhoods combined with the usage of randomized permutation strengthen the exploration to find promising solutions, which are then refined by the local search operations.

If the best known solution obtained so far did not improve after all operations have been tested, the following iteration of the algorithm uses a perturbed version of the best known solution. The perturbation used is based on a geometric distribution. In particular, (uniformly) random numbers r are drawn from $[0, 1]$ and one of the given operations with random parameters is randomly applied on the permutation until r is greater than ps. This distribution favors a small number of perturbations. The strength of perturbation ps increases additively (here set as $\varepsilon = 0.05$ with ps maxed out at 0.99) if successive iterations do not lead to any improvement of the best known solution, otherwise it is reset to the initial value ps_{init}.

3.5 Computational Evaluation

In this section the computational experiments for the Two-Machine Flow Shop with Buffers and their results are described.

3.5.1 Algorithms for Comparison

Based on the literature overview given in Section 3.1 and Figure 3.2 for permutation flow shops with buffers, the *Hybrid Variable Neighborhood Search* (HVNS) and the *Discrete Artificial Bee Colony* (DABC) were selected for a comparison with 2BF-ILS since these algorithms are described in fairly recent works [200, 119] and have not been outperformed by any other algorithm so far. For the implementation, the author of this thesis asked the authors of both algorithms for the source code, but did not receive a reply. Thus, both algorithms were reimplemented. The source code for all algorithms (written in R and C++) and the code for the evaluations in the following sections are available at `https://github.com/L-HT/TwoMachineFlowShopBuffers`.

3.5.2 Generation of Problem Instances

The commonly used benchmark instances for flow shop problems (from [168], [140] and the recent VRF benchmark by [180]) work with at least five machines and contain no buffer constraints so that they cannot be directly used for the flow shops with buffers considered here. Authors that studied flow shop problems with buffers and two machines (e.g., [88] and [107]) generated instances with random processing times uniformly drawn over the set $\{1, 2, \ldots, 100\}$. Based on the studies performed by the these authors, instances for the experiments were generated as follows.

The values $n \in \{50, 100, 150\}$ were chosen as the number of jobs for small, medium and large instances, and three "incomplete instances" were created for each size that contain only the M_1 processing times a_J drawn randomly from a uniform distribution over the set $\{1, 2, \ldots, 100\}$. Each of these incomplete instances was then used to build three subordinate instances by choosing the constant processing times c on M_2 as the median, i.e., the 50 % percentile $q_{0.50}$ of the values a_J. Afterwards, this value of c was used to create (complete) instances for the intermediate buffer (spanning buffer) by setting $s_J = 1$ and $\Omega = 1$ ($\Omega = 3$) or $s_J = a_J$ and $\Omega = q_{0.25}$ ($\Omega = \max a_J + q_{0.25}$) for all jobs J where $q_{0.25}$ is the 25 % percentile of the

values a_J. Note that the buffer capacity Ω cannot be smaller than $\max_J s_J$ for the spanning buffer model or else there exists no feasible schedule. The resulting set of instances contains 36 flow shop problems.

In addition, an additional set of instances was generated based on studies conducted by [8] and [193] where it was argued that problem data generated from uniform distributions do not contain characteristics of problem instances commonly occurring in practical applications, namely gradients or correlations with respect to job processing times. In instances with the former property, the processing times a_J on M_1 are shorter (or longer) than the processing time b_J on the second machine for all jobs J. Regarding flow shops satisfying $b_J = c$ for all jobs J, job gradients lead to the cases analyzed in Section 3.2.1 which were shown to be solvable in polynomial time. The latter characteristic (correlations between processing times) was incorporated by drawing the processing times a_J for M_1 from normal distributions $N(\mu, \sigma^2)$ with mean value $\mu = c$. Higher values for c then lead to tendentially longer processing times on M_1 and vice versa. In particular, n integer values were calculated for processing times a_J by drawing n ($n \in \{50, 100, 150\}$) random numbers $r_1, r_2, \ldots, r_n$ from normal distributions $N(\mu, \sigma^2)$ with $\mu = c$, $\sigma = 10$ and $c = 50$, and setting $a_{J_i} = \max\{\lceil r_i \rceil, 1\}$ for all $i \in \{1, 2, \ldots, n\}$. The values for s_{J_i} are calculated in the same way as for the first set of instances (with uniformly random processing times) leading to 36 additional flow shop problems.

3.5.3 Parameter Values

For 2BF-ILS,the parameter values were determined with the algorithm configurator `irace` [112]. The configuration was performed on a separate set of instances with 100 jobs (generated using the method described in Subsection 3.5.2 for uniformly distributed processing times) with the standard `irace` parameter values and a budget of 250 runs for each algorithm. As possible sequences for operations all sequences of length 1, 2 or 3 were considered that can be formed by the operations *insert*, *pairInsert* or *swap* (without repetition leading to 15 possible sequences in total). The algorithms HVNS and DABC were used in the experiments with the parameters given by the respective authors as well as tuned parameters calculated by `irace` on the same instances. In the following, the tuned versions of these algorithms are referred to as HVNS-T and DABC-T.

Table 3.2: Results for the configuration of numerical parameters for 2BF-ILS, HVNS and DABC as calculated by `irace`, including the upper and lower limits used during the configuration. The parameter names for HVNS and DABC are based on [200, 119].

Algorithm	Parameter	Limits	Result	Description
2BF-ILS	ps_{init}	$[0.01, 0.99]$	0.24	initial perturbation strength
HVNS	N_{iter}	$[1000, 250000]$	135456	number of iterations
	T_{fin}	$[0.010, 0.990]$	0.876	final temperature
DABC	N_S	$[1, 100]$	7	population size
	p	$[1, 10]$	7	perturbation strength
	d_1	$[1, 10]$	7	destruction size 1
	d_2	$[1, 10]$	4	destruction size 2

The values for the numerical parameters calculated by `irace` are given in Table 3.2. Regarding the categorical parameter $\underline{op}$ (the sequence of local search operations used by 2BF-ILS), the sequence $(pairInsert, insert, swap)$ was obtained from the results in `irace`.

After the configuration, each algorithm was executed on all 72 instances and the resulting values were averaged over 30 replications. All test runs were performed on a computer cluster of Leipzig University with thirty-six 2.1-GHz-cores (each run being executed on one core) and 36 GB RAM with time limits of 5, 10 and 15 minutes for the small, medium and large problems, respectively.

3.5.4 Comparison of 2BF-ILS with other Metaheuristics

The performance of each algorithm was evaluated based on two time measures: i) the number of performed *function evaluations* FE to calculate the makespan and ii) the elapsed absolute run time. The latter was used to calculate the *normalized run time* NT as a time measure where the reference run time for a given instance was chosen as the mean run time of 30 runs of the standard NEH heuristic.

The evaluation methodology is based on a benchmarking study by Weise et al. [195] for the Traveling Salesman Problem: In particular, the *progress curve* (PC, the quality of the best known solution over time) and the *empirical cumulative distribution function* (ECDF, describing the percentage of runs reaching a "target

solution quality" over time) were calculated for each instance and time measure. The calculations were performed with respect to the relative percentage difference $RPD = (F(\pi) - \hat{F})/\hat{F}$ between the solution quality $F(\pi)$ of the permutation π on an instance and the best solution quality $\hat{F}$ found in all runs on the same instance. The target value for the ECDF was chosen as $\hat{F}$, i.e., the ECDF diagrams show how consistent an algorithm reaches the best solution quality $\hat{F}$ during the 30 runs on an instance.

Since an individual evaluation of over 250 diagrams is not feasible, the *area under curve* (AUC) was calculated as an aggregate quality measure for the PC and ECDF diagrams (with respect to both time measures) since algorithms with low AUC values (for PC curves) and high AUC values (for ECDF diagrams) tend to find better solutions faster [195]. As such, these values quantify the performance of an algorithm for a given instance over time. In Table 3.3 (for PC diagrams) and Table 3.4 (for ECDF diagrams), relative AUC values are shown for both sets of instances (uniformly distributed processing times and normally distributed processing times). The values are averaged over different subsets in order to analyze how an instance's properties affects the performance of the algorithms.

It can be seen that 2BF-ILS outperforms the other algorithms in all evaluation measures on both sets of flow shop problems as it always obtains the values closest to 1. For progress curves with $\mathsf{tm} = FE$, it even reaches 1.000 which means that it obtains the best performance on all instances. The lowest number is reached for the evaluation measure $ECDF$ and $\mathsf{tm} = FE$ when instances with uniformly random processing times have a spanning buffer or $s_J = a_J$ holds (0.660 and 0.666, respectively). This indicates that these instances are harder for 2BF-ILS and that the algorithm's performance is not as consistent as on instances with normally distributed processing times.

The `irace`-tuned versions of HVNS and DABC obtain mixed results when compared with the original versions. Regarding the progress curves (PC), HVNS-T obtains slightly better results on instances with normal processing times, whereas for DABC this is the case for most of the progress curves with $\mathsf{tm} = FE$ on instances with uniformly random processing times. For HVNS and HVNS-T, small improvements can be observed on progress curves for instances with normally distributed processing times, whereas for these instance no difference was observed with respect to the ECDF curves.

Table 3.3: Relative area under curve (AUC) values for PC diagrams which were calculated for each algorithm as well as each time measure and averaged over different subsets of instances (indicated in the first column). For each instance, the area under the curve was calculated for each diagram and the best obtained value was chosen as the "reference" to calculate the relative values for the other algorithms. The upper half shows the values for the instances with uniformly distributed processed times on M_1, whereas the lower half shows the values for instances with normally distributed processing times. For the progress curves (PC), small values are preferable. A number close to 1 indicates that the curves for the respective algorithm show a high similarity (on average) to the best performing algorithm on each instance. The values closest to 1 are shown in bold for each criterion. "ILS" is used as an abbreviation for 2BF-ILS.

Progress curves (uniform)	tm $= FE$					tm $= NT$				
	HVNS	HVNS-T	DABC	DABC-T	ILS	HVNS	HVNS-T	DABC	DABC-T	ILS
Buffer type										
intermediate	2.777	3.000	2.138	2.194	**1.000**	3.500	3.474	1.916	2.250	**1.004**
spanning	2.288	2.488	2.159	1.896	**1.000**	3.017	3.141	1.766	2.075	**1.000**
Buffer usage										
$s_J = 1$	1.277	1.333	1.555	1.500	**1.000**	2.444	2.277	1.555	1.611	**1.000**
$s_J = a_J$	3.788	4.155	2.742	2.590	**1.000**	4.072	4.338	2.127	2.714	**1.134**
All instances	2.533	2.744	2.149	2.045	**1.000**	3.258	3.307	1.841	2.162	**1.002**
Progress curves (normal)	tm $= FE$					tm $= NT$				
	HVNS	HVNS-T	DABC	DABC-T	ILS	HVNS	HVNS-T	DABC	DABC-T	ILS
Buffer type										
intermediate	2.388	2.333	1.555	1.611	**1.000**	2.309	2.385	1.388	1.777	**1.000**
spanning	1.888	1.777	2.333	2.444	**1.000**	1.944	1.833	2.228	2.679	**1.131**
Buffer usage										
$s_J = 1$	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
$s_J = a_J$	3.277	3.111	2.888	3.055	**1.000**	3.254	3.219	2.617	3.457	**1.131**
All instances	2.138	2.055	1.944	2.027	**1.000**	2.127	2.109	1.808	2.228	**1.065**

When looking at the values averaged over all instances it can also be observed for all algorithms that the values in Table 3.3 and Table 3.4 for the instances with normally distributed processing times are closer to 1 than for uniformly random instances for all evaluation criteria, which indicates that these methods can obtain "good" solutions for instances with processing times a_J similar to c more easily than for uniformly randomized instances.

In addition, for most instances with $s_J = 1$ (for both buffer types and both sets of instances) it was observed that the algorithms quickly reach a state of convergence

Table 3.4: Relative area under curve (AUC) values for ECDF curves, averaged over different sets of instances. The upper half shows the values for the instances with uniformly distributed processed times on M_1, whereas the lower half shows the values for instances with normally distributed processing times. Note that for ECDF curves, higher values are preferable. A number close to 1 indicates that the curves for the respective algorithm show a high similarity (on average) to the best performing algorithm on each instance. The highest values are shown in bold for each criterion. "ILS" is used as an abbreviation for 2BF-ILS.

ECDF (uniform)	tm $= FE$					tm $= NT$				
	HVNS	HVNS-T	DABC	DABC-T	ILS	HVNS	HVNS-T	DABC	DABC-T	ILS
Buffer type										
intermediate	0.448	0.439	0.491	0.471	**0.993**	0.454	0.444	0.553	0.516	**0.993**
spanning	0.415	0.422	0.420	0.432	**0.660**	0.409	0.417	0.446	0.459	**0.826**
Buffer usage										
$s_J = 1$	0.833	0.840	0.839	0.870	**0.987**	0.832	0.839	0.900	0.937	**0.987**
$s_J = a_J$	0.030	0.021	0.073	0.033	**0.666**	0.031	0.022	0.099	0.038	**0.833**
All instances	0.431	0.430	0.456	0.451	**0.827**	0.432	0.430	0.500	0.488	**0.910**
ECDF (normal)	tm $= FE$					tm $= NT$				
	HVNS	HVNS-T	DABC	DABC-T	ILS	HVNS	HVNS-T	DABC	DABC-T	ILS
Buffer type										
intermediate	0.990	0.990	0.947	0.947	**0.999**	0.994	0.994	0.986	0.986	**0.999**
spanning	0.992	0.992	0.941	0.941	**1.000**	0.995	0.995	0.979	0.979	**0.999**
Buffer usage										
$s_J = 1$	0.993	0.993	0.947	0.947	**1.000**	0.995	0.996	0.983	0.983	**1.000**
$s_J = a_J$	0.989	0.989	0.941	0.941	**0.999**	0.993	0.993	0.982	0.982	**0.999**
All instances	0.991	0.991	0.944	0.944	**0.999**	0.994	0.994	0.983	0.983	**0.999**

with the same solution quality (see Figure 3.11 left for an example). A comparison with the lower bound given by the Johnson algorithm [73] for flow shops without buffer constraints showed that in most cases an optimal solution was reached. This can also be seen in Table 3.3 where for $s_J = 1$ values closer to 1 are reached by all algorithms. This effect is even stronger when looking at the values in Table 3.3 for instances with normally distributed processing times where all algorithms obtained the value 1.000. Further experiments using additional instances of this type lead to similar results which indicates that this special case could be "easier" to solve than other problems (even though in theory it is still *NP*-complete). This was not observed for instances with $s_J = a_J$ where a slower convergence occurred in most instances for both sets (see Figure 3.11 right for an example).

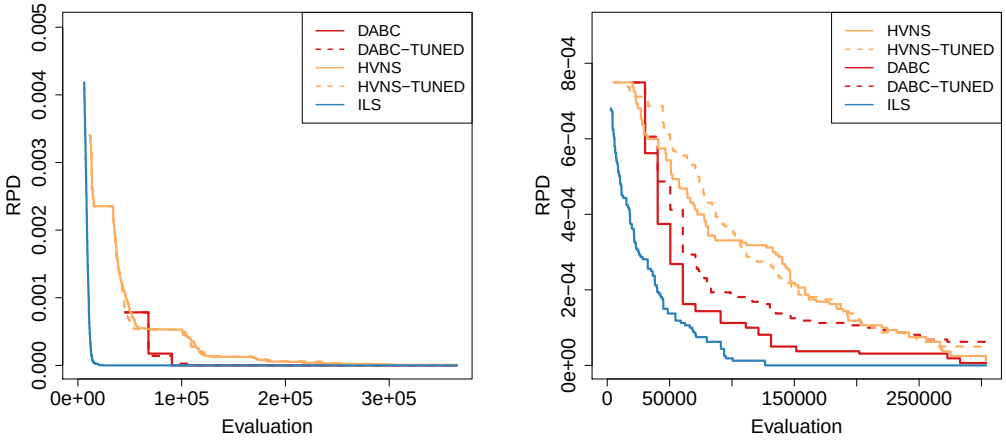


Figure 3.11: Progress curve for an instance using the spanning buffer, $s_J = 1$, uniformly random processing times and 150 jobs (left) and an instance with spanning buffer, $s_J = a_J$ for all J, normally distributed processing times and 100 jobs (right)

From the PC diagrams it was also observed that 2BF-ILS calculated the first feasible solution earlier than the other compared algorithms (note, e.g., how the curves for 2BF-ILS in Figure 3.11 start slightly earlier than the other algorithms). A possible explanation for this is that 2BF-ILS uses the modified NEH (mNEH) described in Section 3.3 which uses fewer function evaluations than the standard NEH algorithm. This indicates that the modified NEH is also suited for cases where a "good" initial solution needs to be quickly obtained.

To compare the performance of the algorithms at specific points in time, the sign test for paired samples was applied. This non-parametric test neither requires the given data to be normally distributed nor the difference distributions between the methods to be symmetric. Using this test, the performance of the algorithms during the run (at 100 000 evaluations) and the performance at the end of the time limit was compared. The results are shown in Table 3.5 for both sets of instances. Note that the tables are symmetric since the 10 possible pairwise comparisons for each of the two points in time were performed with two-sided tests.

Similar to Tables 3.3 and 3.4, it can be observed that 2BF-ILS obtains a significantly better performance than the other algorithms at both time points in almost

all cases indicating a high and consistent performance over time. For instances with uniformly distributed processing times, that the DABC algorithms obtain better results than the HVNS algorithms regarding the quality of the final solution. In the case for normally distributed processing times, the tests showed fewer statistically significant differences (i.e., p-values $p < 0.05/10$) between the competitor algorithms than for instances with uniformly random processing times. A possible explanation for this is that with normally distributed processing times the problem instances becomes "easier" so that more of the competitor algorithms obtain good results, and thus more similar results when compared to 2BF-ILS.

Table 3.5: Results of the pairwise comparisons between the algorithms using the two-sided sign test for both sets of flow shop problems (each containing $n = 36$ instances). The first value in each cell shows the test result with respect to the performance at 100 000 evaluations and the second value refers to the performance reached at the end of the time limit. A triangle indicates that the measured difference is statistically significant ($p < 0.05/10$ due to Bonferroni correction) and that the algorithm at which the triangle is pointed at is significantly better according to the test statistic.

		HVNS	HVNS-T	DABC	DABC-T	2BF-ILS
uniform	HVNS		- / -	- /▲	- /▲	▲/▲
	HVNS-T	- / -		- /▲	- /▲	▲/▲
	DABC	- /◀	- /◀		- / -	▲/▲
	DABC-T	- /◀	- /◀	- / -		▲/▲
	2BF-ILS	◀/◀	◀/◀	◀/◀	◀/◀	
normal	HVNS		- / -	- / -	- / -	▲/▲
	HVNS-T	- / -		- / -	- / -	▲/▲
	DABC	- / -	- / -		- / -	▲/ -
	DABC-T	- / -	- / -	- / -		▲/▲
	2BF-ILS	◀/◀	◀/◀	◀/ -	◀/◀	

Plotting the RPD values of the final solutions for each algorithm on each instance in relation to 2BF-ILS as points $(RPD, RPD_{\text{2BF-ILS}})$ due to the paired nature of the measured data (as shown in Figure 3.12) also shows the high performance of 2BF-ILS. Especially for HVNS (and HVNS-T) on instances with uniformly random processing times it can be seen that for larger problem sizes the gap in solution quality to 2BF-ILS increases. Similarly, for instances with normally distributed

processing times the higher RPD values are measured for larger instances with 100 or 150 jobs. Most of the points in both diagrams are below the diagonal which indicates that even if 2BF-ILS does not obtain the lowest RPD value the obtained makespan is still lower than the makespans of the other algorithms.

In addition, Figure 3.12 right shows that the majority of points are gathered around lower RPD values for 2BF-ILS, HVNS(-T) and DABC(-T) indicating that these algorithms tend to obtain solutions with higher and more similar quality for instances with correlations between the processing times a_i and c than for problems with uniformly random processing times. This observation as well as the differences between the two sets of instances noted above for Tables 3.3 and 3.4 support the results from [8] and [193] stating that instances containing structural features are easier to solve than arbitrarily random problems.

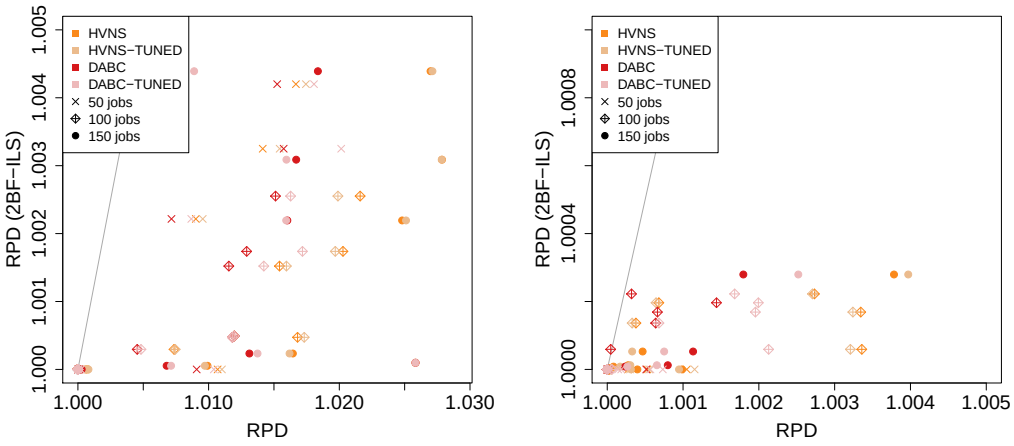


Figure 3.12: Scatter plot of RPD values of the final solution calculated by 2BF-ILS in relation to the respective RPD values of the other algorithms for flow shop instances with uniformly distributed processing times (left) and normally distributed processing times (right) on M_1 for each problem size. The grey line marks the diagonal line $y = x$ such that points above (below) the line indicate that the algorithm obtained a better (worse) final solution than 2BF-ILS.

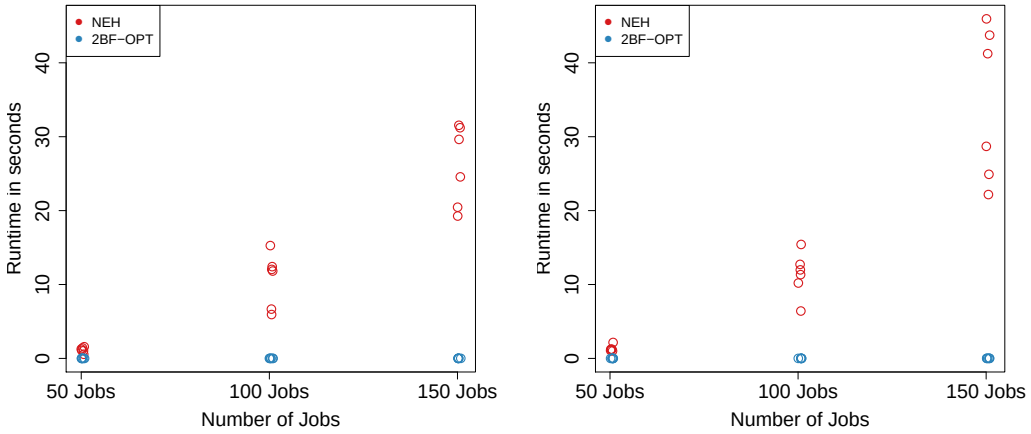


Figure 3.13: Distribution of run times between the standard NEH heuristic and 2BF-OPT on instances of the type $F2|prmu, b_i = c, spanningBuffer, s_i = a_i|C_{\max}$ for instances with $c \leq a_J$ (left) and $c \geq a_J$ (right) for all J. Each point corresponds to the run time on one instance averaged over 30 replications.

3.5.5 Comparison of 2BF-OPT with NEH

As noted in Section 3.3 and Section 3.2.1, both the standard NEH heuristic and 2BF-OPT (Algorithm 3.1) optimally solve instances of the type $F2|prmu, b_i = c, spanningBuffer, s_i = a_i|C_{\max}$ with $c \geq \max_J a_J$ and with $c \leq \min_J a_J$ in polynomial time. In this section, the run time performance of both algorithms is empirically investigated.

Experiments were conducted using the subset of all instances described in Section 3.5.2 that belong to $F2|prmu, b_J = c, spanningBuffer, s_J = a_J|C_{\max}$. For these instances, the constant c was redefined as either $c = \max_J a_J$ or $c = \min_J a_J$ in order to generate suitable test problems (36 instances in total). For both algorithms the run times were measured on each instance and averaged over 30 repetitions. In Figure 3.13, it can be seen that the average run time of NEH grows in an approximately quadratic manner with increasing number of jobs taking significantly more time than 2BF-OPT on all instances ($p < 0.001$, $n = 36$ according to the sign test).

3.6 Summary

In this chapter of the thesis, a special case of the Two-Stage VRP with Profits and Buffers was investigated where no travel times are imposed on the edges and all jobs must be processed. This problem corresponds to a "Two-Machine Flow Shop with Buffer" with the additional restriction that all processing times on the second machine are equal to a constant c.

Flow shop problems with buffers are actively researched in the literature and the theoretical results in Section 3.2 show that they are *NP*-complete even with the restriction to constant processing times on the second machine. However, efficiently solvable subcases were identified where the constant c is smaller or larger than all processing times on the first machine. The most interesting of these subcases is the spanning buffer when c is larger than all processing times on M_1 and where $s_J = a_J$ holds for all jobs J. For this case an algorithm 2BF-OPT was presented which calculates optimal schedules in $O(n \log n)$ steps. In addition to the computational complexity of the problem, conditions that guarantee the existence of permutation schedules in the set of optimal schedules were presented as well as instances where the restriction to permutation schedules leads to a loss in the attainable solution quality by a constant factor when compared to optimal schedules. However, it was shown that in general this factor is bounded from above by 2.

Furthermore, a heuristic mNEH was introduced that is specifically adapted to Two-Machine Flow Shops with constant processing times on the second machine. In particular, it uses this property to reduce the number of function evaluations. Finally, an Iterated Local Search 2BF-ILS was developed for permutation flow shops with buffers that uses mNEH to calculate the initial solution. The algorithm outperforms other state-of-the-art algorithms for permutation flow shops with buffers (namely, a Hybrid Variable Neighborhoods Search and a Discrete Artificial Bee Colony algorithm) with respect to multiple time measures and evaluation criteria. The results also show that the compared algorithms perform better if the flow shop instances contain correlations between processing times, a structural feature that commonly occurs in practical applications.

4 The Special Case $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ Without Processing Times: The Orienteering Problem

The previous chapter considered the special case $\mathcal{O}_{\min C_{\max},\, R \geq \sum r_{J_i}}$ of the Two-Stage VRP with Profits and Buffers where all edges have no travel time. In this chapter, a special case of $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ (i.e., "maximize the collected profit, but the makespan must not exceed B") is investigated where edges have travel times, but for all nodes v the processing times a_{J_v}, b_{J_v} of the corresponding jobs J_v are equal to zero. This can be interpreted in the sense that processing a job (or "servicing a customer at a node") takes a negligible amount of time that is not considered in the optimization problem so that the total time of a route only depends on the time used by the machines M_1, M_2 (or vehicles) to travel on edges of the graph. In this chapter, this "special case of of $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ where the processing times for all jobs on both machines is zero" is abbreviated as $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$.

In order to visualize this special case of the Two-Stage VRP with Profits and Buffers, consider Figure 4.1 which shows an example instance and an example schedule σ. The makespan of σ is $C_{\max}(\sigma) = 15$ and the total profit is $R(\sigma) = 10$. Note that the processed jobs in this figure are indicated above (for M_1) and below (for M_2) the line since all job processing times are zero. In the case where the buffer usage s_J satisfies $s_J = a_J$ for all jobs J, no job takes up any buffer space ($s_J = 0$). The example in Figure 4.1 depicts the case where $s_J = 1$ holds for all J.

For the special case of the Two-Stage VRP with Profits and Buffers with these restrictions, the resulting problem is very similar to the so-called **Orienteering Problem** which can be briefly described as follows: Given a graph $G = (V, E)$ where the edges $e_{ij} \in E$ have weights d_{ij} and the nodes $v \in V$ have values r_v ("profits"), the Orienteering Problem is to find a route that starts at a given depot

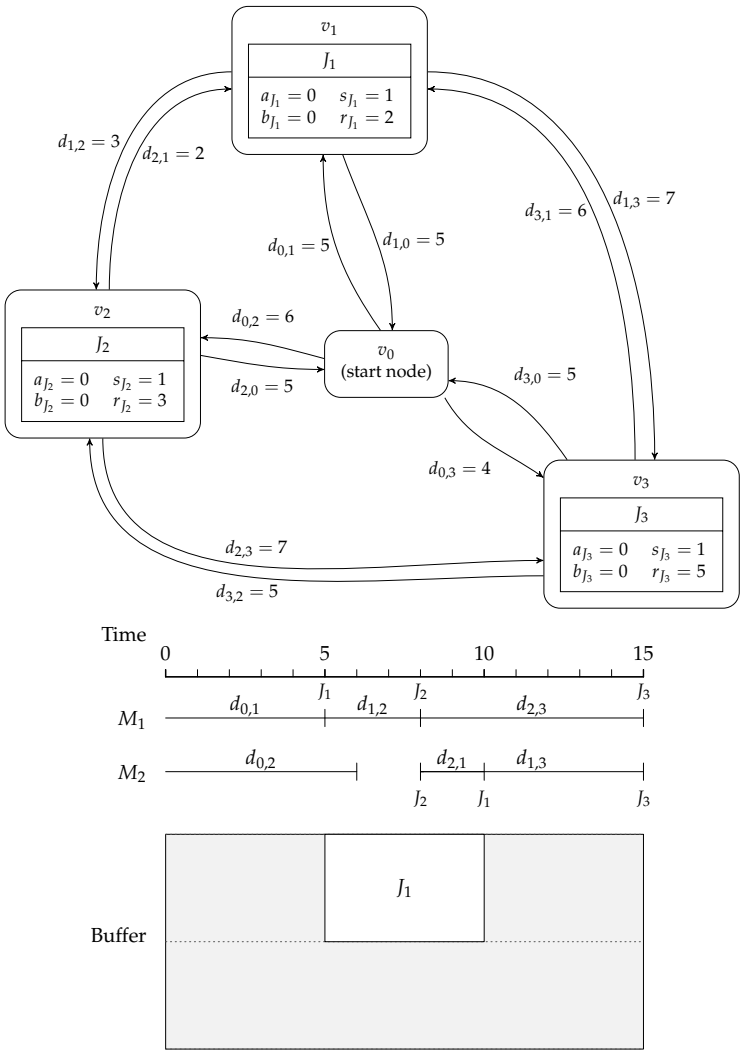


Figure 4.1: Top: Example instance for the Two-Stage VRP with Profits and Buffer- swith the additional property that processing times for all jobs are zero. Bottom: Example for a schedule σ with the paths $P^1(\sigma) = (v_0, v_1, v_2, v_3)$, $P^2(\sigma) = (v_0, v_2, v_1, v_3)$ and the job permutations $\pi^1(\sigma) = (J_1, J_2, J_3)$, $\pi^2(\sigma) = (J_2, J_1, J_3)$. For these paths and job permutations with buffer capacity $\Omega = 2$, the resulting schedule is the same for both buffer models (intermediate buffer and spanning buffer). Note that the names of the processed jobs are indicated above (for M_1) or below the line (for M_2) since all processing times are zero.

node $v_0 \in V$ where the weight sum of the traversed edges does not exceed a given threshold B ("budget") while maximizing the total profit of the visited nodes. Note that not all nodes in G have to be visited.

In fact, it can be shown that the Orienteering Problem is, in a certain sense, equivalent to $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$ under some specific conditions, although some analysis regarding the buffers and non-permutation schedules is necessary in order to see this. This is further discussed in the theoretical analysis performed in this chapter (see Section 4.2). In the following, it is assumed that the graph G in the Orienteering is a directed, complete and simple graph, similar to the graphs assumed for the Two-Stage VRP with Profits and Buffers.

The Orienteering Problem combines aspects of the well-known Traveling Salesperson Problem (TSP) and the Knapsack Problem (KP). The connection to the latter problem lies in the sub-problem of selecting a suitable set of nodes with a high profit where a higher number of visited nodes increases the likelihood that the solution length solution exceeds the budget, whereas the sub-problem of calculating of a short route through the selected nodes is similar to the former problem. However, if the solution calculated for the second sub-problem exceeds the available budget, the first sub-problem might need to be resolved. In other words, solving the Orienteering Problem requires one to take the interplay of these two combinatorial optimization problems with different target criteria into account.

Due to this, this optimization problem is actively researched in the literature and relevant for many practical applications. An overview of the literature on Orienteering Problems and their applications is given in Section 4.1. The aforementioned theoretical analysis regarding the properties of the Orienteering Problem and $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$ is done in Section 4.2. Similar to the analysis on Two-Machine Flow Shops with Buffers in Section 3.2, the computational complexity, the existence of "permutation schedules" (where both machines M_1, M_2 visit the nodes in the same order) in the set of optimal solutions and the existence of gaps between non-permutation schedules and permutation schedules regarding the attainable solution quality are investigated. In Section 4.3, a Variable Neighborhood Search VNS_{OP} is proposed for this problem that is empirically compared with other state-of-the-art methods for the Orienteering Problem in Section 4.4.

This chapter is based on published research from the author, in particular [97, 98].

4.1 Review of Literature on Orienteering Problems

The name "Orienteering Problem" comes from a sport with the same name, where players need to navigate to various check points as quick as possible by means of a compass and a map [179]. The underlying combinatorial optimization problem has applications in many areas, e.g., in the planning of city trips [182] and fuel delivery routes [59], in agriculture [113], or for planning surveillance activities in military scenarios [192]. In many of the applications where the OP occurs, the nodes correspond to locations in a city or a region and the calculated solution corresponds to a route on a road network.

One of the earliest works dealing with the "Orienteering Problem" (OP) is [179] where the name for the problem was first introduced based on the similarities to the sport with the same name. Other names for the OP are Selective TSP [91] or Maximum Collection Problem [77]. There exist several possible formulations of the OP as a linear program (see, e.g., [91, 75]).

The OP is known to be NP-hard [59] and various heuristics for calculating approximate solutions have been developed for this problem. One of them is the Greedy Randomized Adaptive Search Procedure (GRASP) which has multiple variants, such as Memetic GRASP [114] or GRASP with path relinking [29]. The latter is outperformed by a newer GRASP heuristic that removes path segments [79] as well as by an Evolutionary Algorithm [84]. Other heuristics include an Ant Colony Optimization (ACO) for a multi-objective version of the OP [151], an ACO combined with machine learning [166], an Evolution-inspired algorithm with Hill Climbing [127] or an approximation algorithm for the case where the underlying directed graph has unit values for all nodes [120].

An extension of the Orienteering Problem that is actively researched is the **Team Orienteering Problem** where multiple routes need to be calculated that each satisfy the budget constraint, with the profit being summed over all routes, whereas for the Orienteering Problem in its standard formulation only one route is considered. The multiple routes can be interpreted as routes for multiple vehicles which is why the Team Orienteering Problem is sometimes also referred to as a "Vehicle Routing Problem with Profits" [13].

One of the first works that considers this problem is [31] where a constructive heuristic is proposed. Since then, a large number of other heuristic algorithms and

metaheuristics have been developed for this problem, including search algorithms, e.g., a Greedy Randomized Adaptive Search Procedure (GRASP) [160], Tabu Search [169] and a Harmony Search algorithm [178]. A Large Neighborhood Search that incorporates the solution of Set Packing Problems to improve routes is presented in [67]. Kim, Li, and Johnson [82] also propose a Large Neighborhood Search where it is combined with three "improvement algorithms", including Local Search, shifting and replacement.

Regarding biologically inspired algorithms, an Ant Colony Optimization combined with several construction methods for candidate solutions is proposed in [78], and a Genetic Algorithm is presented [47] in the context of waste collection and recycling. An interesting example is Particle Swarm Optimization which is originally a metaheuristic for continuous optimization problems, but it is successfully applied to the Team Orienteering Problem in [5, 36] by combining tours of multiple particles to "giant tours".

Combinations of methods can also be found in the literature regarding the Team Orienteering Problem. For example, a Path Relinking metaheuristic is combined with a GRASP algorithm in [159], and in [23] a Memetic Algorithm is developed that combines a Genetic Algorithm with Local Search. Due to the large number of proposed heuristics and metaheuristics, Mısır, Gunawan, and Vansteenwegen [118] propose a method based on Collaborative Filtering for selecting a suitable algorithm for the Team Orienteering Problem. Their approach obtains promising results, but the performance is significantly affected by the available time budget.

Exact methods have also been proposed for the Team Orienteering Problem, for example, Branch and Bound [20, 37] as well as a Branch-and-Price algorithm [24] combined with Dynamic Programming [80]. Xu et al. [196] present an approximation algorithm with a parameter ε that for $\varepsilon = 0.5$ has a guaranteed approximation ratio of at least 0.32.

Extensions and Related Problems

An increasing number of research works focus on extensions of Orienteering Problems, i.e., problems with additional components which are not based on the Orienteering sport, but arise in practical scenarios that are structurally similar to Orienteering Problems.

One way to extend the Orienteering Problem is to introduce probabilistic elements. The resulting problems are referred to as "Stochastic Orienteering Problems". In one variant [9] it is possible that nodes randomly become unavailable while traversing a route so that some of the nodes have to be skipped. In this problem, the difference between the expected total value of the path and the expected path length is to be maximized. To solve this problem, the authors of [9] propose a Mixed-Integer Linear model and a metaheuristic. In another variant [71] the node values are randomized with normally distributed values and the probability that the total value of a path exceeds a given target value is to be maximized. For this problem an exact algorithm as well as a Genetic Algorithm are presented in [71]. In [201] the sale of books in the context of a university campus is presented as a Stochastic OP. This problem is modelled as a Markov Decision Process and approximately solved using an Approximate Dynamic Programming algorithm while allowing the route to dynamically change while the path is traversed.

Campbell, Gendreau, and Thomas [28] introduce a Variable Neighborhood Search algorithm for the Orienteering Problem with stochastic travel and service times that achieves results similar to a Dynamic Programming approach but is significantly faster at computing solutions. Papapanagiotou, Montemanni, and Gambardella [132] also propose a Variable Neighborhood Search for the Stochastic Orienteering Problem and expand their approach by incorporating a sampling-based evaluator for the objective function. Solving the Stochastic Orienteering Problem is also important in robotics since stochastic travel times can be used to model uncertain terrain. For example, the approach of Thayer and Carpin [174] calculates for each vertex where an irrigation robot should go next in a vineyard while taking a battery constraint into account.

Another extension of Orienteering Problems are "Dynamic Orienteering Problems" where components of the problem change over time. For example, so-called time windows, i.e., time intervals are given that represent business hours at which certain nodes have to be visited (see, e.g., the surveys in [64, 56] or the application to the city of Tehran in [1]). This problem is known as "Orienteering Problem with Time Windows". A Dynamic Orienteering Problem where the profits and the budget change over time (with "time" referring to the runtime of an algorithm) has been considered by the author in [97, 98]. Another interpretation for time in "change over time" is the time when a node or an edge is visited, which primarily

depends on the distance previously traveled on a given path. This case is studied in [48] where the weights of the edges (representing travel times) dynamically change depending on the departure time from the starting node. Another variant where node values linearly decrease depending on when nodes are visited is studied by Erkut and Zhang [41].

The Orienteering Problem combines aspects of the well-known Traveling Salesperson Problem (TSP) and the Knapsack Problem (KP), so it is also relevant to look at works dealing with dynamic variants of these problems. The TSP has been extensively investigated in the literature and various studies deal with dynamic versions of the TSP. Dynamic changes in the TSP can be the removal or addition of vertices as well as changes in the distance between vertices or in the traversal times assigned to the edges. Schmitt, Baldo, and Parpinelli [152] propose an Ant System with a short-term memory for the dynamic TSP. Chowdhury et al. [32] also modify an ACO framework with an Adaptive Large Neighborhood Search to generate new solutions for the dynamic TSP by destroying and repairing fractions of the current solution. In [164] a Particle Swarm Optimization algorithm is developed for the same problem, which is competitive with two ACOs. The survey of Feillet, Dejax, and Gendreau [45] gives an overview of other variants of Traveling Salesperson Problems incorporating profits, but these problems are structurally different from the Orienteering Problem as traveling on the edges on their variants also incurs a cost that reduces the profit.

The Dynamic Knapsack Problem deals, e.g., with a changing capacity of the knapsack. Roostapour, Neumann, and Neumann [143] introduce single- and multi-objective Evolutionary Algorithms to account for the changing knapsack capacity and show that the multi-objective approach outperforms the single-objective approach. The authors strengthen their results and expand their method in [144]. Assimi et al. [14] also use single- and multi-objective Evolutionary Algorithms to deal with a knapsack with changing capacity as well as stochastic and unknown item weights. Their results state that bi-objective optimization already outperforms single-objective optimization.

There exist other variants of the Orienteering Problem with additional components, which are not related to the problems considered in this thesis. An overview of these variants is given in the surveys [64, 183, 74]. The survey of Gavalas, Konstantopoulos, and Pantziou [56] focuses on connections to the related, but structurally

different Tourist Trip Design Problem. The review of Martins et al. [116] considers Team Orienteering Problems (and related Vehicle Routing Problems) in the context of electric cars with limited battery capacity and their environmental impact.

4.2 Theoretical Properties

In this section the relationship between the Orienteering Problem and $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ is analyzed. In particular, it is shown that under certain conditions these two problems are equivalent. In order to show this, some preliminary analyses regarding theoretical properties for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ are needed which deal with the existence of permutation schedules in the set of optimal solutions, potential gaps in solution quality between permutation schedules and non-permutation schedules and the computational complexity of the problem, similar to the properties analyzed for the Two-Machine Flow Shop Problem with Buffers in Section 3.2.

Regarding non-permutation schedules for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$, the following lemma holds.

Lemma 4.2.1. *Let σ be a non-permutation schedule for an instance of $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$. Then, there exists a permutation schedule $\hat{\sigma}$ that satisfies $VisitedNodes(\sigma) = VisitedNodes(\hat{\sigma})$, $R(\sigma) = R(\hat{\sigma})$ and $C_{\max}(\sigma) = C_{\max}(\hat{\sigma})$.*

Proof. Note that the length of σ is determined by when M_2 finishes the last job. Construct $\hat{\sigma}$ by having M_2 take the same path as in σ, i.e., $P^2(\hat{\sigma}) = P^2(\sigma)$, and setting all start and completion times on M_2 in $\hat{\sigma}$ to be identical to the times in σ: $S^2_J(\hat{\sigma}) = S^2_J(\sigma)$ and $C^2_J(\hat{\sigma}) = C^2_J(\sigma)$ for all $J \in ProcessedJobs(\sigma)$. Next, the same times and path are "copied" to M_1, i.e., $S^1_J(\hat{\sigma}) = S^2_J(\hat{\sigma})$ and $C^1_J(\hat{\sigma}) = C^2_J(\hat{\sigma})$ for all jobs J and $P^1(\hat{\sigma}) = P^2(\hat{\sigma})$. Since travel times d_{ij} are the same for both machines and all edges e_{ij}, it is possible for M_1 to reach all nodes in $P^1(\hat{\sigma})$ in time and process the corresponding jobs (with zero processing time). Since $C^1_J(\hat{\sigma}) = S^2_J(\hat{\sigma})$, it is also possible for M_2 to start the processing of J at the specified times so that $\hat{\sigma}$ is a valid schedule.

Regarding the buffer, note that the buffer is not used in the intermediate buffer, whereas for the case with spanning buffer the amount of used buffer capacity does not exceed the highest amount of used buffer in the original schedule σ at any point in time, so $\hat{\sigma}$ is also feasible. By definition, $VisitedNodes(\sigma) = VisitedNodes(\hat{\sigma})$

and $R(\sigma) = R(\hat{\sigma})$ hold so that $\hat{\sigma}$ is the desired schedule satisfying $C_{\max}(\sigma) = C_{\max}(\hat{\sigma})$. □

The previous lemma describes a procedure to transform a non-permutation schedule σ into a permutation schedule $\hat{\sigma}$ with the same length. The following lemma details how a permutation schedule can be transformed into a permutation schedule where all jobs are started as early as possible.

Lemma 4.2.2. *Let $\hat{\sigma}$ be a permutation schedule for an instance of $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$. Let $P(\hat{\sigma}) = (v_0, v_{i_1}, v_{i_2}, \ldots, v_{i_\ell})$ be the sequence of nodes visited by M_1 and M_2 in $\hat{\sigma}$. Then, there exists a permutation schedule σ^P that satisfies VisitedNodes(σ^P) = VisitedNodes$(\hat{\sigma})$, $R(\sigma^P) = R(\hat{\sigma})$ and for all $k \in \{1, 2, \ldots, \ell\}$ it holds that*

$$S^1_{J_{v_k}}(\sigma^P) = C^1_{J_{v_k}}(\sigma^P) = S^2_{J_{v_k}}(\sigma^P) = C^2_{J_{v_k}}(\sigma^P) = \begin{cases} d_{0,i_1} & (k = 1) \\ d_{0,i_1} + \sum_{t=1}^{k-1} d_{i_t,i_{t+1}} & (k > 1), \end{cases}$$

which means that the starting and completion times for all jobs J_{v_k} are the sum of travel times starting from the depot node v_0 up to the node v_k on the path $P(\hat{\sigma})$. In addition, $C_{\max}(\sigma^P) \leq C_{\max}(\hat{\sigma})$.

Proof. The following procedure is used to transform a permutation schedule $\hat{\sigma}$ into a permutation schedule σ^P with the desired properties. Without loss of generality, it is assumed that $P(\hat{\sigma}) = (v_0, v_1, v_2, \ldots, v_\ell)$ and $\pi(\hat{\sigma}) = (J_1, J_2, \ldots, J_\ell)$ (this is possible by renaming the nodes and their corresponding jobs).

Note that it is not possible that $S^1_{J_1}(\hat{\sigma}) < d_{0,1}$. If $S^1_{J_1}(\hat{\sigma}) > d_{0,1}$, this implies that M_1 has idle times so that the processing of J_1 can be moved to the left so that $S^1_{J_1}(\sigma^P) = d_{0,1}$ holds. The same can be done for $S^2_{J_1}(\sigma^P)$ if $S^2_{J_1}(\hat{\sigma}) > d_{0,1}$ without affecting the validity of σ^P since $S^1_{J_1}(\sigma^P) = C^1_{J_1}(\sigma^P) \leq S^2_{J_1}(\sigma^P)$ holds by construction. Next, consider J_2. The earliest possible time for J_2 to start on M_1 and M_2 in σ^P is $C^1_{J_1}(\sigma^P) + d_{1,2} = d_{0,1} + d_{1,2}$, so J_2 is scheduled to start at that time on M_1 and M_2. With similar arguments as above for J_1, the validity of the schedule is not affected, i.e., it is possible for M_1 and M_2 to start the processing J_2 at this time. The jobs $J_3, J_4, \ldots, J_\ell$ can be inductively scheduled with similar arguments which leads to σ^P satisfying the desired properties. Finally, note that the lower bound $C_{\max}(\hat{\sigma}) \geq d_{0,1} + d_{1,2} + \cdots + d_{\ell-1,\ell}$ holds for the original schedule $\hat{\sigma}$. Since σ^P has exactly that length, the inequality $C_{\max}(\sigma^P) \leq C_{\max}(\hat{\sigma})$ follows. □

In the following, the schedule σ^P constructed from the permutation schedule $\hat{\sigma}$ by the procedure in Lemma 4.2.2 is referred to as the "minimal permutation schedule" (to $\hat{\sigma}$) as it is the shortest possible permutation schedule with the same order as in $\hat{\sigma}$ where all jobs are processed as early as possible. It also has the property that it uses the buffer as little as possible: For intermediate buffers, the buffer is not used at all, whereas for spanning buffers not more than $\max_{J \in ProcessedJobs(\hat{\sigma})} s_J$ units of buffer are used. Recalling the common choices for buffer usage values s_J, in the case $s_J = a_J$ for all J this means that $s_J = 0$ so that no buffer capacity is used. For the case $s_J = 1$ for all J, not more than 1 buffer unit is used at any time, which is the lowest possible value for the buffer capacity Ω (or else no feasible schedules would exist in the spanning buffer case). For this reason, it can be assumed in the following that all minimal permutation schedules do not violate the buffer constraint.

The two procedures described in Lemma 4.2.1 and Lemma 4.2.2 can be applied to any non-permutation schedule (or permutation schedule) to obtain a minimal permutation schedule without a loss in solution quality with respect to $R(\sigma^P)$ and $C_{\max}(\sigma^P)$. For this reason, the following analyses for $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$ can be restricted to only minimal permutation schedules.

Now, it is shown that $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$ is equivalent to the Orienteering Problem.

Theorem 4.2.1. *For a given instance I_{OP} of the Orienteering Problem, there exists an instance I^* of the optimization problem $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$ where for all solutions σ_{OP} of I_{OP} there exists a minimal permutation schedule σ^* in I^* with the same length and the same total profit. Furthermore, for a given instance I_0 of $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$, there exists an instance of the Orienteering Problem I^*_{OP} where for all minimal permutation schedules σ_0 of I_0 there exists a solution σ^*_{OP} in I^*_{OP} with the same length and the same total profit.*

Proof. For the first statement, the instance I^* is constructed from I_{OP} as follows. The instance I^* uses the same graph, the same depot node v_0 and the same edge weights. For each node v the corresponding job J_v is defined to have zero processing times for both machines and the same profit as the corresponding node in I_{OP}. The buffer type in I^* is assumed to be appropriately chosen so that all minimal permutation schedules are feasible (for example, by using an intermediate buffer). Let σ_{OP} be a solution for I_{OP} that, without loss of generality, is assumed to traverse the nodes $v_0, v_1, v_2, \ldots, v_\ell$ in that order. This solution has the total length $d_{0,1} + d_{1,2} + \cdots + d_{\ell-1,\ell}$ and collects the total profit $r_1 + r_2 + \cdots + r_\ell$. The corresponding

solution σ^* of I^* is set to be a minimal permutation schedule that traverses the same (corresponding) nodes in the same order so that it has the same length as σ_{OP} and the same total profit.

For the second statement, the instance I^*_{OP} of the Orienteering Problem is constructed from I_0 as follows. The instance I^*_{OP} uses the same graph, the same depot node v_0 and the same edge weights. For each node v and corresponding job J_v in I_0 with profit r_{J_v}, the corresponding node in I^*_{OP} is set to have the same profit. Let σ_0 be a minimal permutation schedule for I_0. As argued above, it can be assumed that σ_0 is feasible with respect to the buffer constraint (or else no feasible schedules would exist in I_0). The corresponding solution σ^*_{OP} for I_{OP} traverses the same (corresponding) nodes in the same order, so that it has the same total length and the same total profit as σ_0. □

Computational Complexity

With the statement shown above, it is now possible to analyze the computational complexity for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$.

Theorem 4.2.2. *The decision problem whether for a given instance I_0 of $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ and two non-negative numbers B and Q there exists a solution σ with $C_{\max}(\sigma) \leq B$ (budget constraint) and $R(\sigma) \geq Q$ (minimum score constraint) is NP-complete.*

Proof. The problem $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ is in *NP* since it can be verified in polynomial time whether a given solution satisfies the budget constraint and the minimum score constraint.

In order to show *NP*-hardness, a straightforward reduction from the Orienteering Problem, which is known to be *NP*-complete [59], is used. The decision version of the Orienteering Problem is whether for a given Orienteering Problem instance with budget B' and non-negative number Q' there exists a path that collects a total profit of at least Q' without exceeding the budget B'. The reduction from the Orienteering Problem is done analogous to the proof of Theorem 4.2.1 with $B = B'$ and $Q = Q'$. Let I_{OP} be an Orienteering Problem instance and I^* be the instance of $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$. It is straightforward to see that the existence of a solution in I_{OP} that satisfies the profit and length requirements implies the existence of an equivalent solution in I^* that is a minimal permutation schedule.

For the inverse direction, assume that a solution σ exists in I^* that satisfies the budget constraint and the minimum score constraint (while not necessarily being a permutation schedule or a minimal permutation schedule). Using Lemma 4.2.1 and Lemma 4.2.2, a minimal permutation schedule σ^* can be constructed from σ that also satisfies both constraints, after which σ^* can be converted to a solution of the Orienteering Problem (with the same procedure as in the proof of Theorem 4.2.1) that collects the required profit Q' without exceeding the budget B'. Thus, a solution for one problem that satisfies the length and profit requirements exists if and only if a solution for the other problem with the required properties exists. □

The original *NP*-hardness proof for the Orienteering Problem by Golden, Levy, and Vohra [59] uses a reduction from the Traveling Salesperson Problem (TSP) so that one might think that the hardness of the Orienteering Problem stems from the TSP-like sub-problem of calculating a short route through a subset of nodes. However, it can be shown that the Orienteering Problem is still an *NP*-hard problem even if the travel lengths d_{ij} in the graph are chosen such that a shortest path through any subset of nodes can be efficiently calculated.

This is due to the other sub-problem that is similar to the Knapsack problem where a set of nodes has to be chosen such that their total value is maximized without violating a weight constraint (or in the Orienteering Problem, the budget constraint). This can be seen by outlining an alternative *NP*-hardness proof that uses a reduction from the Knapsack problem: Assume that n items $x_1, x_2, \ldots, x_n$ are given with values r_i, weights w_i ($i \in \{1, \ldots, n\}$) and a weight limit W for the Knapsack problem. The corresponding Orienteering Problem instance uses $n + 1$ nodes $v_0, v_1, \ldots, v_n$ with the same profit values r_i (an exception is the depot node v_0 with $r_0 = 0$ which acts as a dummy node), the budget $B = W$ and the travel lengths $d_{i,j} = w_j$ for all edges e_{ij} in the graph (with $d_{i,0} = 0$, if $j = 0$).

It is not hard to see in the constructed instance of the Orienteering Problem that for a given subset of nodes $S \subseteq V$ all routes (that do not contain unnecessary detours) have the same length. A consequence of this is that shortest paths through a set of nodes can be efficiently calculated. Furthermore, it can be seen that any valid solution in the Knapsack problem has an equivalent and valid solution in the constructed instance of the Orienteering Problem with the same total value, and vice versa (if unnecessary detours are excluded).

Inversely, simplifying only the Knapsack-like component of the Orienteering Problem (for example, by assuming that all nodes v have the value $r_v = 1$) still leaves the TSP-like sub-problem that needs to be dealt with. This reinforces the notion that the Orienteering Problem combines aspects of two *NP*-hard combinatorial optimization problems so that algorithms for this problem need to take both sub-problems into account.

Optimal Permutation Schedules for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\ C_{\max}\leq B}$

Similar to the theoretical analysis done for the Two-Machine Flow Shop Problem with Buffers in Section 3.2, it is also possible to investigate the existence of permutation schedules in the set of "optimal solutions" for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\ C_{\max}\leq B}$. Recall that for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\ C_{\max}\leq B}$ the optimization criterion is the profit that is to be maximized so that the term "optimal" refers to solutions that maximize the collected profit without exceeding the budget. In order to investigate the existence of permutation schedules with this property, Lemma 4.2.1 can be directly used on any non-permutation schedule (including optimal non-permutation schedules) to show the following statement.

Corollary 4.2.1. *For every instance of $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\ C_{\max}\leq B}$, the set of optimal schedules always contains a permutation schedule.*

This result shows that the restriction to permutation schedules for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\ C_{\max}\leq B}$ does not restrict the attainable solution quality, i.e., there is no "gap" between schedules in general (including non-permutation schedules) and permutation schedules. This can be formalized as follows.

Corollary 4.2.2. *For a given instance of $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\ C_{\max}\leq B}$, let $R(\sigma^*_{perm})$ be the highest possible profit that can be attained with a feasible permutation schedule (without violating the budget constraint) an a given instance. Let $R(\sigma^*)$ be the highest possible profit that can be obtained with any schedule (including non-permutation schedules) on the same instance without exceeding the budget. Then, the solution quality ratio $R(\sigma^*)/R(\sigma^*_{perm})$ satisfies $R(\sigma^*)/R(\sigma^*_{perm}) = 1$.*

4.3 A Variable Neighborhood Search for the Orienteering Problem

Since the Orienteering Problem is known to be *NP*-hard [59], several metaheuristics have already been proposed for this problem (see Section 4.1). In this section, a Variable Neighborhood Search is presented for the Orienteering Problem (and due to Theorem 4.2.1, also for $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$).

Variable Neighborhood Search (VNS) has been successfully applied to some extensions of the Orienteering Problem (see Section 4.1) as well as other routing problems related to dynamic optimization. For example, Sarasola et al. [150] propose a VNS to solve a Dynamic Vehicle Routing Problem in which a fleet of vehicles need to supply several customers at minimal cost and new customers may be added that need to be supplied. Their VNS outperforms an Ant Colony Optimization algorithm, a Genetic Algorithm, and a Tabu Search algorithm if one considers the best over all runs. Khouadjia et al. [81] compared a VNS with a Particle Swarm Optimization (PSO) algorithm and found that the VNS computes better solutions on average and outperforms the PSO on larger instances.

The core principle of Neighborhood Search algorithms is to modify a given solution σ_{old} by selecting a new solution σ_{new} from a set (the "neighborhood" of σ_{old}) containing solutions similar to σ_{old} while maximizing or minimizing an objective function f. Variable Neighborhood Search algorithms are based on the idea that a local maximum or local minimum within one neighborhood might not be a local optimum within a different neighborhood [68]. For this reason, Variable Neighborhood Search algorithms use different neighborhood functions, i.e., sets of solutions considered "similar to σ_{old}" over the course of the algorithm. Formally, this means that new solutions σ_{new} are calculated as $\sigma_{new} = \arg\min_{\sigma \in N_k(\sigma_{old})} f(\sigma)$ (in the case of a minimization problem), where $N_k(\sigma_{old})$ is the neighborhood of solution σ_{old} and k is an index to denote different neighborhoods.

For the proposed VNS algorithm, two neighborhoods $N_{\mathsf{add}}(\sigma)$ and $N_{\mathsf{remove}}(\sigma)$ are used. The set $N_{\mathsf{add}}(\sigma)$ is the set of all solutions σ' derived from σ by inserting one unvisited node into σ (regardless of whether the length of the new solution exceeds the budget B). The set $N_{\mathsf{remove}}(\sigma)$ is defined to be the set of all paths obtained by removing a non-depot node from σ. In other words, search steps with respect to these neighborhoods allow the algorithm to add and remove nodes.

Since solving Orienteering problems requires an algorithm to repeatedly solve sub-problems with different objectives, the heuristic proposed in this work is based on a generalization of this principle: Given a path σ_{old}, the algorithm selects a new solution σ_{new} from a neighborhood N_k that maximizes or minimizes an objective function f_ℓ with an index ℓ which also changes during the run time of the algorithm:

$$\sigma_{new} = \begin{cases} \arg\max_{\sigma \in N_k(\sigma_{old})} f_\ell(\sigma, \sigma_{old}) & \text{if } f_\ell \text{ is to be maximized} \\ \arg\min_{\sigma \in N_k(\sigma_{old})} f_\ell(\sigma, \sigma_{old}) & \text{if } f_\ell \text{ is to be minimized} \end{cases} \tag{4.1}$$

The combination of different neighborhoods with changing objective functions can also be interpreted in the sense that the neighborhood structure is now characterized by the pair (N_k, f_ℓ), which allows a more fine-grained tuning of the algorithm's behavior. More specifically, different objective functions f_ℓ allow the algorithm to focus on specific aspects of the given optimization problem. Since the Orienteering Problem is equivalent to the special case $\mathcal{O}^{a_J=0,\, b_J=0}_{\max R,\, C_{\max} \leq B}$ of the Two-Stage VRP with Profits and Buffers (as was shown in Theorem 4.2.1), the notation $R(\sigma)$ and $C_{\max}(\sigma)$ is used to refer to the total profit and the total length of a path σ, respectively. Based on this notation, the following objective functions are used:

$$f_{\mathsf{length}}(\sigma, \sigma_{old}) = 1/|C_{\max}(\sigma) - C_{\max}(\sigma_{old})| \tag{4.2}$$

$$f_{\mathsf{value}}(\sigma, \sigma_{old}) = |R(\sigma) - R(\sigma_{old})| \tag{4.3}$$

$$f_{\mathsf{ratio}}(\sigma, \sigma_{old}) = \left| \frac{R(\sigma) - R(\sigma_{old})}{C_{\max}(\sigma) - C_{\max}(\sigma_{old})} \right| = f_{\mathsf{length}}(\sigma, \sigma_{old}) \cdot f_{\mathsf{value}}(\sigma, \sigma_{old}) \tag{4.4}$$

$$f_{\mathsf{random}}(\sigma, \sigma_{old}) = r \tag{4.5}$$

In f_{random} the number r is drawn randomly from a uniform distribution over the interval $[0, 1]$. For this case it can be shown that Equation (4.1) is equivalent to choosing σ_{new} randomly from the set $N_k(\sigma_{old})$. In the following, $F = \{f_{\mathsf{length}}, f_{\mathsf{value}}, f_{\mathsf{ratio}}, f_{\mathsf{random}}\}$ is defined to be the set of considered objective functions. If $C_{\max}(\sigma) = C_{\max}(\sigma_{old})$, it is defined that $f_{\mathsf{length}}(\sigma, \sigma_{old}) = \infty$ and $f_{\mathsf{ratio}}(\sigma, \sigma_{old}) = \infty$.

Depending on the objective function f_ℓ and the considered neighborhood (i.e., N_{remove} or N_{add}), the objective function must be minimized or maximized. The absolute values are used to consider only positive values for the objective function f_ℓ. For example, if one considers the function f_{length} with the neighborhood N_{add},

$C_{\max}(\sigma) \geq C_{\max}(\sigma_{old})$ holds and thus f_{length} needs to be maximized, since it is preferable for the new solution σ to not be much longer than σ_{old}. If one considers f_{length} and the neighborhood N_{remove}, $C_{\max}(\sigma) \leq C_{\max}(\sigma_{old})$ holds and thus f_{length} must be minimized to ensure a large difference between the path length of the new and the old solution. In general, for f_ℓ with $\ell \in \{\text{length}, \text{value}, \text{ratio}\}$, the following holds:

$$f_\ell(\sigma, \sigma_{old}) \text{ is to be } \begin{cases} \text{maximized} & \text{if one considers } N_{\text{add}} \\ \text{minimized} & \text{if one considers } N_{\text{remove}} \end{cases} \tag{4.6}$$

The choice of objective functions is motivated by the structure of the Orienteering Problem. For example, f_{length} focuses on the path length $C_{\max}(\sigma)$ and prefers short paths which is relevant for the aforementioned sub-problem that corresponds to the TSP. The objective function f_{value} primarily considers the total value $R(\sigma)$ of a path without taking its length into account which can be interpreted as solely focusing on the Knapsack sub-problem. The third objective function f_{ratio} combines f_{length} and f_{value} such that the selection of new solutions takes both sub-problems into account. Function f_{random} corresponds to a random selection of paths which adds a perturbation mechanism to the algorithm and strengthens the exploration. For the neighborhoods considered in this work, two neighboring solutions differ by a single node so that the calculation of these functions can be performed efficiently by only considering the differing nodes.

The proposed algorithm VNS_{OP} is shown in Algorithm 4.1. VNS_{OP} starts with a short, exploration-focused **initial phase** containing a randomized procedure, which, for a path σ and parameter $p \in [0, 1]$, iterates over all unvisited nodes and inserts them into σ at random positions with probability p, regardless of whether the length of the resulting path violates the budget constraint. This exploration-focused procedure is used to do a quick, cursory scan of different areas of the solution space in order to find a promising subset of nodes from which solutions are then improved using the Chained Lin-Kernighan heuristic [10]. The implementation of this heuristic is based on the implementation provided by the Concorde TSP Solver [33] and its computation time is also included in the time measurement framework that is later described in Section 4.4. Afterwards and if necessary, nodes are removed while minimizing f_{ratio}. The reason for using this objective function is that it provides a balanced view on both sub-problems without inducing any

Algorithm 4.1 VNS_{OP}

Input: initial iterations k_{init}, initial insertion probability p

$\sigma \leftarrow$ empty solution (that only contains the depot node v_0)
for $it = 1, \ldots, k_{init}$ **do** ▷ initial exploration phase
    for each node v not in σ **do**
        with probability p, insert v into σ at a random position
    end for
    optimize the order of nodes in σ using the Chained Lin-Kernighan heuristic
    while $C_{\max}(\sigma) > B$ **do**
        $\sigma \leftarrow \arg\min_{\sigma' \in N_{\mathsf{remove}}(\sigma)} f_{\mathsf{ratio}}(\sigma', \sigma)$
    end while
end for
while termination criterion not satisfied **do** ▷ main iterations
    $f_1, f_2 \leftarrow$ random elements from the set F
    repeat
        $\sigma \leftarrow \arg\max_{\sigma' \in N_{\mathsf{add}}(\sigma)} f_1(\sigma', \sigma)$
    until $C_{\max}(\sigma) > B \vee N_{\mathsf{add}}(\sigma) = \varnothing$
    optimize the order of nodes in σ using the Chained Lin-Kernighan heuristic
    while $C_{\max}(\sigma) > B$ **do**
        $\sigma \leftarrow \arg\min_{\sigma' \in N_{\mathsf{remove}}(\sigma)} f_2(\sigma', \sigma)$
    end while
end while
return best solution found so far

significant computational overhead. The initial phase lasts for k_{init} iterations where k_{init} is a parameter.

Afterwards, the **main iterations** start where the algorithm loops through three steps until a termination criterion is satisfied. With the solution σ obtained from the previous step, it repeatedly performs search steps using neighborhood N_{add} while maximizing a randomly chosen function from F until the resulting path exceeds budget B. The obtained path is now invalid. Next, the calculated path σ is improved with respect to $C_{\max}(\sigma)$ by using the Chained-Lin-Kernighan heuristic. This heuristic is used in the algorithm since it is used in the Concorde TSP Solver that has been successfully applied to the TSP and derived problems.

In the third step, if the improved path σ still violates the budget constraint, the algorithm performs search steps with respect to $N_{\mathsf{remove}}(\sigma)$ in order to remove nodes

from σ. In the VNS framework, this corresponds to a change in neighborhood since a local optimum with respect to $N_{\mathsf{add}}(\sigma)$ was reached that cannot be feasibly improved by adding nodes. These search steps minimize a randomly chosen function from F until the length of the resulting path σ does not exceed the budget B anymore. The random selection of functions is done in order to provide variations in each iteration of the algorithm, which in combination with the function f_{random} strengthens the exploration and allows the algorithm to constantly test a variety of paths based on the current solution.

4.4 Computational Evaluation

In this section, the performance of the proposed algorithm VNS_{OP} is evaluated.

4.4.1 Measurement of Algorithm Performance

When measuring an optimization algorithm's performance, it is not sufficient to consider an algorithm's performance only at a certain point in time [195] as this neglects the algorithm's optimization behavior over time, in particular its convergence speed. Rather, the algorithm's optimization behavior over a given time interval needs to be taken into account. For this reason, a measurement framework with logging functionalities has been implemented (with the source code being available at [95]). In order to measure how well an algorithm performs over time, **progress curves** (PC) are used based on [195], which plot over time the quality $R(\sigma_{best})$ of the best solution σ_{best} found so far. In addition, Empirical Cumulative Distribution Functions (ECDFs) are calculated which describe over time the percentage of runs that reach a target solution quality $\hat{R}$. For the following experiment, the target solution quality $\hat{R}$ for an instance is chosen to be a deviation that is less than 1 % of the best solution quality R^* found on that instance, i.e., a value R is said to have reached the target quality if it satisfies $(R^* - R)/R^* \leq 1\,\%$.

Similar to the study performed by Weise et al. [195], progress curves and ECDFs are recorded with respect to different time measures tm in order to evaluate different aspects of an algorithm. Due to their general definition, they can be easily used for other optimization problems if appropriate problem-specific time measures are chosen. In this work, the following three time measures tm are used:

1. Function evaluations (FE) count how often the total length $C_{\max}(\sigma)$ or the total value $R(\sigma)$ of σ is calculated. Evaluations with respect to this time measure give insight into how an algorithm deals with the TSP sub-problem since for a given subset of nodes $V' \subseteq V$ there exist multiple paths with different lengths.

2. Subsets (SS) count how many different subsets $V' \subseteq V$ of nodes have been used so far for the calculation of paths. This time measure focuses on how an algorithm selects suitable subsets of V and thus how it deals with the sub-problem that is similar to the Knapsack Problem.

3. The normalized time (NT) is the runtime that has passed since the start of the algorithm normalized by a reference run time. This measure can be used to compare the performance of algorithms on different computers with respect to runtime performance. In contrast, the time measures FE and SS do not depend on the hardware that is used to run an algorithm. For this experiment, the reference runtime for a given instance is chosen to be average runtime of the Chained Lin-Kernighan heuristic [10] over 30 runs on that instance.

Since the criterion $R(\sigma)$ is to be maximized for the Orienteering Problem, it is preferable for the resulting progress curve to quickly reach high values as opposed to progress curves for minimization problems such as the TSP or $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ where low values in the target criterion are desirable. This allows one to use the value $UB = \sum_{v \in V} r_v$, i.e., the sum of all node values at a time as a trivial upper bound for normalizing the solution quality such that $R(\sigma)/UB \in [0,1]$ holds. This approach is similar to the "optimization accuracy" measure used for dynamic optimization problems [123]. Example progress curves are shown in Figure 4.2 left for algorithms which are described in the following.

4.4.2 Choice of Algorithms for Comparison

Based on the literature overview given in Section 4.1, the Greedy Randomized Adaptive Search Procedure (GRASP) with Segment Remove from [79] and the Evolutionary Algorithm from [84] were selected as reference algorithms to evaluate the proposed method VNS_{OP}. Various algorithms based on GRASP have been proposed for the Orienteering Problem (see Section 4.1), and the heuristic proposed by Keshtkaran and Ziarati [79] is a fairly recent algorithm which also outperforms

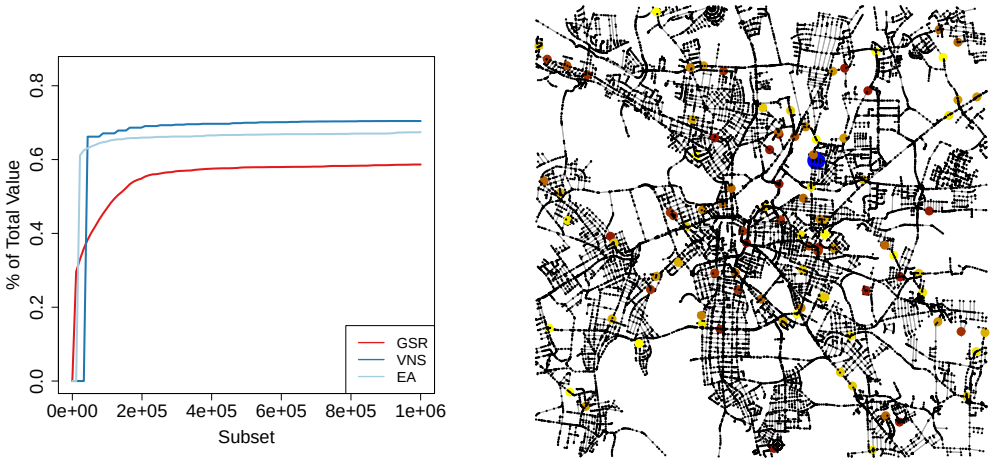


Figure 4.2: Left: Examples for progress curves with respect to time measure tm $= SS$ (Subset) for the Evolutionary Algorithm (EA), the Variable Neighborhood Search (VNS_{OP}, abbreviated as "VNS") and the Greedy Randomized Adaptive Search Procedure with Segment Remove (GSR). Right: Visualization of the Orienteering Problem instance `Leipzig-100-80000-3` generated on basis of OpenStreetMap data for Leipzig. The blue node is the depot v_0 and the other colored nodes correspond to nodes with assigned values where a bright color indicates a high value.

another GRASP that incorporates path relinking. The Evolutionary Algorithm from Kobeaga, Merino, and Lozano [84] also obtained favorable results when experimentally compared with other heuristics, including another GRASP algorithm [29]. In the following, they are abbreviated as GSR and EA, respectively.

The authors of EA uploaded their source code to GitHub, but for the experiments in this work the algorithm has been reimplemented based on the source code and the description in [84] in order to fit with and utilize our measurement framework for the DOP mentioned in Section 4.4.1. Algorithm GSR [79] was also reimplemented as its source code is not available. The source code (in C++) for the algorithms EA and GSR as well as the proposed algorithm VNS_{OP} is available at [95]. In order to make GSR comparable with EA and VNS_{OP} (since it uses a different termination criterion), the algorithm has been slightly modified. In particular, if the local search phase ends before the termination criterion is satisfied, the path obtained so far is modified using the Segment Remove operator proposed in [79] after which the algorithm starts the next iteration with its `localSearch` procedure.

4.4.3 Problem Instances

The instances for the Orienteering Problem used in the following experiments are based on two sets of instances. The first set is based on `OPLib` [85], a benchmark for Orienteering Problems. From this benchmark the instances `brazil58`, `brazil48`, `gr48`, `gr120` were chosen from the `gen4`-subset since it contains the most difficult instances [84, 85]. These instances were also used in the study where EA is proposed [84]. The 4 instances from the subset were chosen because they specify the travel times d_{ij} for the edges using a distance matrix, whereas most of the other instances specify distances by listing point coordinates and calculating the Euclidian distances between them. However, for various applications, e.g., on road networks it is not uncommon that the travel distance between two nodes differs from their Euclidian distance which is why these instances were not considered.

The second set of instances (in the following referred to as `city`) is intended to contain properties of realistic road networks in cities and was generated as follows: Map data from OpenStreetMap [125] was used from which extracts for two German cities, Leipzig and Berlin (as examples for a smaller and a larger city) were downloaded using the download server from [153]. Then, a parser [76] was applied, after which the roads were extracted and processed in order to obtain the road network as a graph. On these two graphs, we randomly selected n nodes (with $n \in \{50, 100, 150\}$) and assigned to them a random value $r_v \in \{1, 2, \ldots, 10\}$ as their "profit[1]" excluding one random node which was set as a depot node v_0 with value 0. This was repeated three times for each city. In addition, distance matrices for these nodes were calculated so that the instance data is available in the same form as in the `OPLib` instances.

Regarding the budget B for the `city` instances, the value $B = 80\,000$ was used based on the following reasoning: A survey [72] measured that the average speed in the two aforementioned cities is 11 mph (≈ 17.70 km/h). Since road transport drivers in the European Union are not allowed to drive for more than 4.5 hours without taking a break [197], it is potentially possible to drive 49.5 miles during that time, which, after rounding, corresponds to approximately 80 000 m since the generated graphs measure distances in meters. Using the Concorde TSP Solver

[1] Since the Orienteering and $\mathcal{O}^{a_j=0,\, b_j=0}_{\max R,\, C_{\max} \leq B}$ were shown to be equivalent (see Theorem 4.2.1), it can also be said from the perspective of the latter problem that r_v is the profit of a job J_v belonging to node v where J_v has no processing times for both machines M_1 and M_2.

[33], it has been verified that for none of the `city` instances all nodes can be reached within this budget. This set of instances contains 18 Orienteering Problems and an example is shown in Figure 4.2 right.

4.4.4 Parameter Values

The proposed algorithm VNS_{OP} described in Section 4.3, contains two parameters, namely the number of iterations in the initial phase k_{init} and the probability p for random insertions in the initial phase. Regarding k_{init}, since the number of possible solutions grows rapidly with increasing number of nodes in the graph, we consider it reasonable to scale the length k_{init} of the initial exploration phase with the size of the graph. However, if the algorithm focuses too strongly on exploring the solution space, there might not be enough time to refine the discovered solutions. Thus, the parameter k_{init} was set as $k_{init} = \sqrt{|V|}$, i.e., the square root of the number of nodes as a compromise between these two conflicting aspects.

As for the second parameter p, it is desirable that p is close to the percentage of nodes contained in an optimal solution so that the insertion procedure and subsequent optimization of the path length lead to a solution of high quality. Kobeaga, Merino, and Lozano [84] dealt with a similar problem for their algorithm EA and proposed the formula $\sqrt{B/C_{\max}(\sigma_{LK})}$, which in the following is also used for the parameter p. This formula incorporates the budget B as well as the length of the path σ_{LK} obtained by applying the Chained Lin-Kernighan heuristic on all nodes in V, so that it provides an efficiently calculable approximation for the number of nodes contained in an optimal solution. The parameters for the other two algorithms EA and GSR are set as described in their respective studies [79, 84].

4.4.5 Experimental Setup

The three compared improvement heuristics were executed on each of 22 Orienteering Problem instances described in Section 4.4.3 with 30 repetitions over which the results were averaged. The runs were executed on a computer cluster of Leipzig University with thirty-six 2.1-GHz-cores (each run being executed on one core) and 36 GB RAM. Similar to the experimental study performed in Section 3.5, the runs were set to terminate after a time limit of 5, 10 and 15 minutes for the small ($n \leq 50$), medium ($50 < n \leq 100$) and larger problems ($n > 100$), respectively.

Plotting the progress curves and ECDFs for each instance and time measure tm leads to a total of 132 diagrams (similar to Figure 4.2 left) so that an individual evaluation of each diagram is not feasible. For this reason, we calculate the percentage that the area under the progress curve (**area under curve**, AUC) occupies from the area of a theoretical progress curve with constant value 1. This value is denoted as AUC^{rel} (**relative AUC**) in the following and satisfies $AUC^{rel} \in [0,1]$ providing an aggregate quality measure for the calculated curves, similar to the evaluation methodology in [195]. These values allow us to quantify the performance of an algorithm on a given instance where a high value indicates that the algorithm quickly obtains solutions of high quality. Note that in contrast to the experiments for the Two-Machine Flow Shop Problem with Buffers in Section 3.5, high values in the progress curves are desirable.

In order to compare the AUC values over different instances (similar to [123]), these values were normalized with respect to the best attained value per instance. Formally, if $AUC^{rel}_{I,A}$ denotes the relative AUC for algorithm A on an instance I, the **normalized value** $AUC^{norm}_{I,A} \in [0,1]$ is calculated as

$$AUC^{norm}_{I,A} = \frac{AUC^{rel}_{I,A}}{\max_{A'} AUC^{rel}_{I,A'}} \tag{4.7}$$

with $A' \in \{\text{VNS}_{\text{OP}}, \text{EA}, \text{GSR}\}$, where a value close to 1 indicates that the algorithm A reaches a performance similar to the best performing algorithm for instance I. This type of evaluation measure can be seen as an extension of the "collective mean fitness" that is commonly used to evaluate an algorithm's performance on dynamic optimization problems [123], where instead of the best values per iteration/generation the optimization behavior over the entire run time is taken into account. An overview of the evaluation metrics used in this section is shown in Table 4.1.

4.4.6 Comparison of VNS_{OP} with other Metaheuristics

In this section, the proposed algorithm VNS_{OP} is compared with EA and GSR with respect to the evaluation measures described in Section 4.4.5. Table 4.2 shows the average values for $AUC^{norm}_{I,A}$ over the progress curves of all instances, as well as average values when only certain sets of instances are considered. These numbers

Table 4.1: Overview of the evaluation metrics used during the computational experiments.

Metric	Description
$AUC_{I,A}$	Area under the progress curve (which shows the solution quality, normalized to $[0,1]$, over time for a given time measure tm) of an algorithm A on an instance I.
$AUC^{rel}_{I,A}$	Ratio between $AUC_{I,A}$ and the AUC of a theoretical progress curve with constant value 1. Satisfies $0 \leq AUC^{rel}_{I,A} \leq 1$.
$AUC^{norm}_{I,A}$	Ratio between $AUC^{rel}_{I,A}$ and AUC^{rel}_{I,A^*} where A^* is the algorithm that obtained the best AUC^{rel} on instance I. Satisfies $0 \leq AUC^{norm}_{I,A} \leq 1$, and if an algorithm A obtains the best results on an instance I, $AUC^{norm}_{I,A} = 1$ holds.

quantify the average optimization performance of an algorithm over time. The values $AUC^{rel}_{I,A}$ used for calculating the averages are uploaded to [95].

It can be seen that VNS_{OP} reaches the best performance for almost all aggregation criteria and time measures. One exception can be found for the time measure $\text{tm} = SS$ (subsets) when only the `oplib` instances are considered as the performance of VNS_{OP} is tied with EA. However, the average AUC values for the remaining criteria indicate that VNS_{OP} has a good performance over time for both sets of instances. For the time measures $\text{tm} = FE$ and $\text{tm} = NT$, it even reaches the value 1.000 which means that it obtained the highest AUC value on all instances used for calculating the average.

It can also be seen for GSR that it performs better for smaller instances than for larger instances. A possible explanation for this is that for a smaller number of nodes, the `addVertex` phase of GSR ends earlier so that more time is spent in the local search phase to improve solutions (see the description of GSR in [79] for details). However, it can also be seen that the gap between EA and VNS_{OP} is smaller than between GSR and VNS_{OP} which indicates that EA reaches a higher performance than GSR. As for GSR, even though it obtains good results for the time measure SS when compared to FE (indicating that it carefully selects a subset $V' \subseteq V$ and thoroughly tests it before changing the subset) it is still outperformed

Table 4.2: Average values for $AUC_{I,A}^{norm}$ for the progress curve diagrams per time measure, aggregated over different subsets of instances. The values are truncated to 3 decimal places and values in bold indicate the best average value for each aggregation criterion and time measure.

Progress Curves	$\mathsf{tm} = FE$			$\mathsf{tm} = SS$			$\mathsf{tm} = NT$		
	EA	GSR	VNS_{OP}	EA	GSR	VNS_{OP}	EA	GSR	VNS_{OP}
Instance set									
`oplib`	0.991	0.910	**1.000**	**0.998**	0.978	**0.998**	0.995	0.978	**1.000**
`city`	0.936	0.807	**1.000**	0.979	0.912	**0.995**	0.968	0.943	**1.000**
Instance size									
$n \leq 50$	0.980	0.958	**1.000**	0.986	0.977	**0.990**	0.989	0.972	**1.000**
$50 < n \leq 100$	0.936	0.821	**1.000**	0.971	0.921	**1.000**	0.969	0.945	**1.000**
$n > 100$	0.917	0.679	**1.000**	0.989	0.867	**0.996**	0.958	0.928	**1.000**
All instances	0.946	0.825	**1.000**	0.982	0.924	**0.995**	0.973	0.949	**1.000**

by VNS_{OP} in all criteria and both time measures. A similar, but smaller difference regarding the time measure $\mathsf{tm} = FE$ in relation to $\mathsf{tm} = SS$ and $\mathsf{tm} = NT$ can also be observed for EA.

Table 4.3: Average values for $AUC_{I,A}^{norm}$ with respect to ECDF diagrams per time measure, aggregated over different subsets of instances. The values are truncated to 3 decimal places and values in bold indicate the best average value for each aggregation criterion and time measure.

ECDF	$\mathsf{tm} = FE$			$\mathsf{tm} = SS$			$\mathsf{tm} = NT$		
	EA	GSR	VNS_{OP}	EA	GSR	VNS_{OP}	EA	GSR	VNS_{OP}
Instance set									
`oplib`	0.719	0.087	**1.000**	0.656	0.223	**0.993**	0.749	0.205	**1.000**
`city`	0.223	0.088	**1.000**	0.354	0.182	**1.000**	0.291	0.164	**1.000**
Instance size									
$n \leq 50$	0.723	0.224	**1.000**	0.720	0.258	**0.998**	0.778	0.262	**1.000**
$50 < n \leq 100$	0.011	0.019	**1.000**	0.013	0.111	**1.000**	0.101	0.135	**1.000**
$n > 100$	0.147	0.001	**1.000**	0.578	0.200	**0.973**	0.186	0.106	**1.000**
All instances	0.313	0.088	**1.000**	0.421	0.191	**0.995**	0.374	0.172	**1.000**

Table 4.3 shows the average $AUC_{I,A}^{norm}$ values with respect to the Empirical Cumulative Distribution Function (ECDF) which describes how consistent an algorithm reaches a certain solution quality. It can be seen that VNS_{OP} obtains the best results for the criteria shown in the table. However, for VNS_{OP} some of the values for the

time measure $\texttt{tm} = SS$ are lower than for the other time measures which indicates that in regards to that time measure the algorithm needs to evaluate a higher number of subsets before the target solution quality (a deviation from the best solution quality found on a given instance that is not more than 1%) is reached.

Similar to Table 4.2, it can be observed for GSR and EA that the values are lower for the instance set `city` than for `oplib` (for all time measures). This means that there is a larger "gap" in how consistent the quality of the calculated solutions for EA and GSR when compared to VNS_{OP} which possibly indicates that VNS_{OP} is better suited for instances containing characteristics of road networks.

It is interesting to note that for medium-sized instances and $\texttt{tm} \in \{SS, NT\}$ GSR outperforms EA. In combination with the data in Table 4.2 (where GSR is outperformed by EA), this indicates for such instances GSR does not always find a better solution than EA, but it reaches a "good" solution quality more consistently than EA when time is measured with respect to these measures. However, for $\texttt{tm} = FE$, the difference is smaller and for larger instances $n > 100$ the value for GSR is close to 0 which means that there is a high number of runs where the target solution quality was not reached within the time limit.

Interestingly, the values for EA and GSR are noticeably small for medium-sized instances when compared to the other instance sizes. A possible explanation for this is that for small instances, all algorithms manage to find high-quality solutions, whereas for large instances the larger search space makes it more difficult for all algorithms (including the best-performing algorithm on an instance) to consistently obtain good solutions. This effect, combined with the normalization with the best-performing algorithm in $AUC_{I,A}^{norm}$, leads to "more similar" values, see Figure 4.3 left for an example. However, for medium-sized instances only VNS_{OP} consistently obtains "good" solutions over time (see Figure 4.3 right for an example) leading to a larger gap when compared to EA and GSR.

But in general it can be seen that the values for large instances ($n > 100$) are smaller than the values for smaller instances ($n \leq 50$). A likely explanation for this is that larger instances are "more difficult" due to the larger solution space so that it is less likely for an algorithm to consistently reach a good solution quality.

In addition, the sign test for paired samples was applied to compare the performance of the algorithms at specific points in time. In particular, the solution quality after 100,000 function evaluations and 100,000 subsets (which can be interpreted as

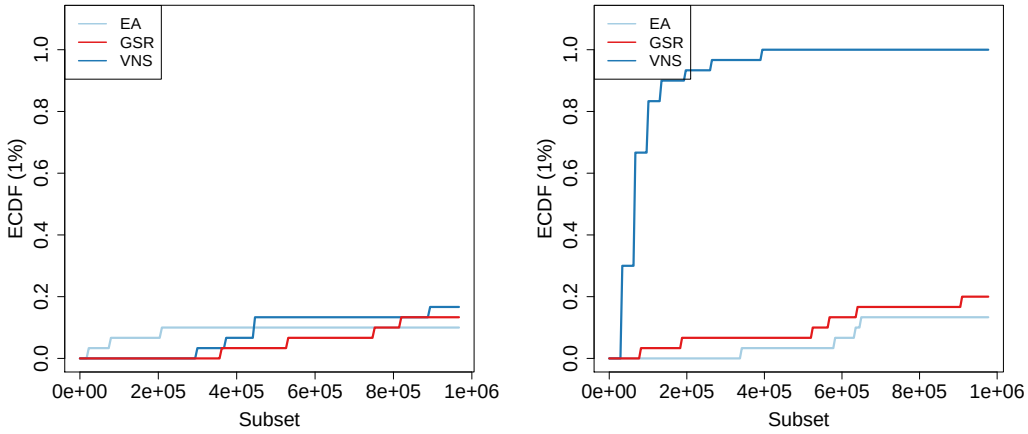


Figure 4.3: Visualization of ECDFs for the instance `LGF-Berlin-150-80000` (left, containing $n = 150$ nodes) and `LGF-Berlin-100-80000-1` (right, containing $n = 100$ nodes) with respect to the time measure $\text{tm} = SS$. The notation "VNS" is used as an abbreviation for VNS_{OP}. Note how on the left, none of the algorithms reach high values, whereas on the right only VNS_{OP} has a high percentage of runs that reach the target solution quality.

the "short-term performance") as well as the quality of the final solution at the end of the run (the "long-term performance") were considered, where the latter is the same as comparing the algorithm performance at the time of termination which was done in [84] and [79].

The results of the statistical tests are shown in Table 4.4. It can be clearly seen that the differences between VNS_{OP} and the algorithms GSR and EA is statistically significant, with VNS_{OP} obtaining better results than the other algorithms. This difference can be visualized by plotting the relative percentage difference $RPD = (R^* - R)/R^*$ for EA and GSR (with R being the quality of the current solution at a given time and R^* being the best solution quality found on an instance over all runs) on each instance in relation to VNS_{OP} as points $(RPD, RPD_{\text{VNS}_{\text{OP}}})$ due to the paired nature of the measured data, as shown in Figure 4.4. Note that in this diagram, values close to 0 are desirable.

On the left, it can be seen that after only 100,000 evaluation there are many points where none of the algorithms have obtained the value $RPD = 0$, especially for larger

Table 4.4: Results of the pairwise comparisons between the algorithms using the two-sided sign test over all Orienteering Problem instances ($n = 22$) at certain points in time. A triangle indicates that the measured difference is statistically significant ($p < 0.05/9$ due to Bonferroni correction) and that the algorithm at which the triangle is pointed at is significantly better according to the test statistic. Note that the tables are symmetric since the 3 possible pairwise comparisons for each of the three points in time were performed with two-sided tests.

	at $t = 100,000\,FE$			at $t = 100,000\,SS$			at end of run		
	EA	GSR	VNS_{OP}	EA	GSR	VNS_{OP}	EA	GSR	VNS_{OP}
EA		◀	▲		◀	▲		◀	▲
GSR	▲		▲	▲		▲	▲		▲
VNS_{OP}	◀	◀		◀	◀		◀	◀	

instances with $n > 100$ which also indicates that these instances are more difficult to solve than smaller instances. However, the different axis scaling in Figure 4.4 right shows that all algorithms have improved their solutions at the end of the runs, but most of the points satisfy $RPD_{\text{VNS}_{\text{OP}}} < RPD$ (i.e., they are below the diagonal $y = x$) which means that at that time VNS_{OP} still obtains better solutions. In particular, there are many points with $RPD_{\text{VNS}_{\text{OP}}} = 0$ which means that at that time VNS_{OP} has already obtained solutions with quality R^*. In addition, the results in Table 4.4 show that GSR is outperformed by EA at the considered points in time. This can also be seen in Figure 4.4 left where many of the points for GSR are positioned to the right of points for EA.

4.5 Summary

In this thesis chapter the Orienteering Problem was considered which is closely related to a special case $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ of the Two-Stage VRP with Profits and Buffers, where the processing times of all jobs on both machines is zero. A theoretical analysis showed that these two problems can be considered to be equivalent. The Orienteering Problem forms a challenging combinatorial optimization problem since it contains aspects from two *NP*-hard problems as sub-problems, namely the Traveling Salesperson Problem and the Knapsack problem. It was shown that simplifying

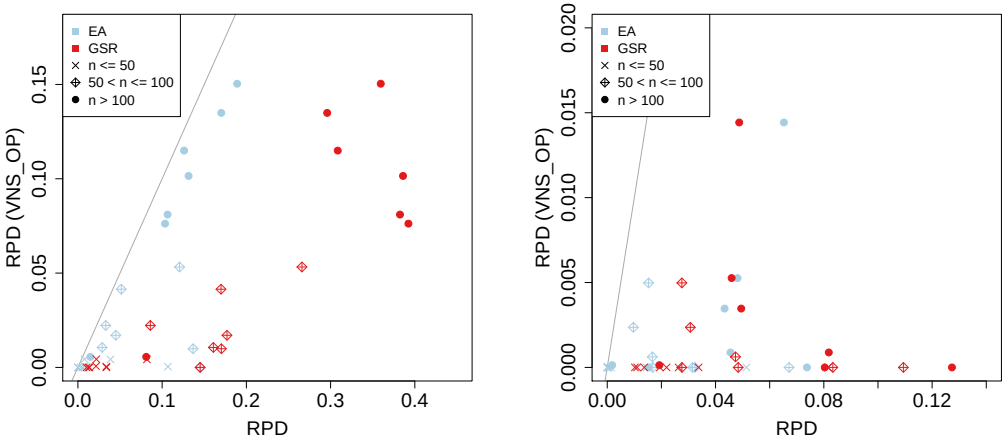


Figure 4.4: Scatter plot of average RPD values for the solution quality of VNS_{OP} at $t = 100,000\,FE$ (left) and at the end of the run (right) in relation to the respective RPD values of the other algorithms. The grey line marks the diagonal line $y = x$ such that points above (below) the line indicate that an algorithm obtained a better (worse) final solution than VNS_{OP}.

one of the sub-problems does not change the *NP*-hardness of the problem. In addition, it was shown for $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ from the perspective of the Two-Stage VRP with Profits and Buffers that the set of optimal schedules always contains at least one permutation schedule so that the restriction to permutation schedules in $\mathcal{O}^{a_J=0,\,b_J=0}_{\max R,\,C_{\max}\leq B}$ does not restrict the attainable solution quality.

Since the Orienteering is known to be *NP*-hard, a heuristic VNS_{OP} was proposed for the Orienteering Problem that is based on Variable Neighborhood Search. The main idea of VNS_{OP} is to take the two interacting sub-problems of the Orienteering Problem into account by combining neighborhoods with variable objective functions. This allows the algorithm to dynamically change how new solutions are generated. This concept can also be applied to other combinatorial optimization problems by considering their characteristics, their structure and appropriately choosing the available functions and neighborhoods.

Computational experiments were performed with two existing state-of-the-art methods for the Orienteering Problem and the proposed algorithm VNS_{OP}. The evaluation of the experiments considered the performance over the entire run time

using graphical evaluation measures, as well as the performance at specific points in time using statistic tests. The results showed that VNS_{OP} outperforms the other algorithms with respect to multiple time measures in most of the considered evaluation criteria. The gap in performance was larger for the set of instances that contain characteristics of road networks when compared to instances from an existing benchmark. In addition, it was observed that larger instances are harder to solve as the compared algorithms took more time and were less consistent at reaching a high solution quality.

5 The Two-Stage Vehicle Routing Problem with Profits and Buffers

After the previous two chapters of this thesis focused on two special cases of the Two-Stage VRP with Profits and Buffers which arise in various practical applications, this chapter considers the Two-Stage VRP with Profits and Buffers in its general case as described in Section 2.2 without any further restrictions. In particular, both optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ are investigated in theoretical analyses and computational experiments. In particular, their theoretical properties regarding computational complexity, existence of optimal permutation schedules and the gap in attainable solution quality between permutation schedules and non-permutation schedules are analyzed in Section 5.1. Afterwards, a metaheuristic framework for the general case of the Two-Stage VRP with Profits and Buffers is proposed in Section 5.2 from which multiple algorithms can be derived for both optimization problems. In Section 5.3, various algorithms derived from the framework are evaluated in computational experiments. The results of this chapter are summarized in Section 5.4.

5.1 Theoretical Properties of the Two-Stage VRP with Profits and Buffers

Similar to the theoretical analyses for the two special cases of the Two-Stage VRP with Profits and Buffers in Section 3.2 and Section 4.2, the computational complexity, the existence of permutation schedules in the set of optimal solutions as well as potential gaps in solution quality between permutation schedules and non-permutation schedules are investigated in the following. Since the Two-Stage VRP with Profits and Buffers contains the Two-Machine Flow Shop with Buffers and the Orienteering, the results presented in this section generalize some of the theoretical

properties shown in Section 3.2 and Section 4.2. In addition, properties for other special cases that can be directly derived from these results are presented in an overview.

5.1.1 Computational Complexity of the General Problem

Since the Two-Machine Flow Shop with Buffers as well as the Orienteering Problem are known to be *NP*-complete (see Section 3.2.1 and Section 4.2) and since the Two-Stage VRP with Profits and Buffers generalizes these two problems, it directly follows that the Two-Stage VRP with Profits and Buffers is also a hard problem.

A formalization of this property is given in the following statement which describes the complexity of the problem's decision variant (for both $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$):

Theorem 5.1.1. *For all values of* bufType $\in \{intermediateBuffer,\ spanningBuffer\}$ *and* bufUsage $\in \{s_J = 1,\ s_J = a_J\}$, *the decision problem whether for a given instance of the Two-Stage VRP with Profits and Buffers and two positive numbers* B, Q *there exists a feasible schedule* σ *with* $C_{\max}(\sigma) \leq B$ *and* $R(\sigma) \geq Q$ *is NP-complete.*

Similar to the analysis in Section 3.2, it might be interesting to investigate special cases of the Two-Stage VRP with Profits and Buffers that can be efficiently solved. This is detailed further below in Section 5.1.4.

5.1.2 Existence of Permutation Schedules in the Set of Optimal Solutions

Another question that was already investigated in the theoretical analyses of the Two-Machine Flow Shop with Buffers and the Orienteering Problem is the question whether the set of optimal schedules contains a permutation schedule. In the following, the same question is analyzed for the Two-Stage VRP with Profits and Buffers. Since both optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ are considered, the following lemma is needed that establishes a connection between their target criteria.

Lemma 5.1.1. *Let* $\mathcal{I}$ *be an instance of the Two-Stage VRP with Profits and Buffers. If, for every non-permutation schedule* σ *for* $\mathcal{I}$ *there exists a permutation schedule* σ^P *with* $ProcessedJobs(\sigma) = ProcessedJobs(\sigma_{perm})$ *and* $C_{\max}(\sigma_{perm}) \leq C_{\max}(\sigma)$, *then the set of*

optimal schedules for instance $\mathcal{I}$ with respect to the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ contains at least one permutation schedule.

Proof. Consider a non-permutation schedule σ^* that is optimal with respect to the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$. Obviously, σ^* has to satisfy $C_{\max}(\sigma^*) \leq B$ or else σ^* would violate the budget constraint. Due to the assumption, there exists a permutation schedule σ^*_{perm} that satisfies $ProcessedJobs(\sigma^*) = ProcessedJobs(\sigma^*_{perm})$ and $C_{\max}(\sigma^*_{perm}) \leq C_{\max}(\sigma^*)$. Due to the latter property, it is not possible that σ^*_{perm} violates the budget constraint. The former property implies that $R(\sigma^*) = R(\sigma^*_{perm})$, so σ^*_{perm} is a permutation schedule that is optimal with respect to $\mathcal{O}_{\max R,\, C_{\max} \leq B}$. □

It is shown in the following that for the Two-Stage VRP with Profits and Buffers, the existence of optimal permutation schedules (with respect to both $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$) is only guaranteed for up to $n = 2$ jobs, which is remarkably small when compared with the bound $n = 6$ that holds for the Two-Machine Flow Shop with spanning buffer and $s_J = a_J$ (see Theorem 3.2.4). This is due to the addition of travel times d_{ij} which allow for a larger class of instances.

Lemma 5.1.2. *Let σ be a non-permutation schedule for an instance of the Two-Stage VRP with Profits and Buffers with $|ProcessedJobs(\sigma)| \leq 2$. Then, there exists a permutation schedule σ^P satisfying $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$, $C_{\max}(\sigma^P) \leq C_{\max}(\sigma)$ and $R(\sigma^P) = R(\sigma)$.*

Proof. If $|ProcessedJobs(\sigma)| = 1$, it is not possible for σ to be a non-permutation schedule, so let σ be a non-permutation schedule where exactly 2 jobs are processed. Without loss of generality, it is assumed that $\pi^1(\sigma) = (J_1, J_2)$ and $\pi^2(\sigma) = (J_2, J_1)$. A trivial lower bound for $C_{\max}(\sigma)$ is $\tilde{C} = d_{0,1} + a_{J_1} + d_{1,2} + a_{J_2} + c + d_{2,1} + c$. If a spanning buffer is used, the buffer capacity Ω must satisfy $\Omega \geq s_{J_1} + s_{J_2}$, whereas for the intermediate buffer the buffer capacity must be at least s_{J_1}.

Next, consider the permutation schedule σ^P that processes the jobs J_1, J_2 in that order on both machines as early as possible. Obviously, it satisfies $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$ and $R(\sigma^P) = R(\sigma)$. By considering the different buffer types with the established lower bounds for Ω, it is not hard to see that regardless of the buffer type, σ^P has the makespan $C_{\max}(\sigma^P) = d_{0,1} + a_{J_1} + \max\{d_{1,2} + a_{J_2} + c,\ c + d_{1,2} + c\}$. This value cannot be larger than the lower bound $\tilde{C}$ for $C_{\max}(\sigma)$. Thus, σ^P is the permutation schedule with the desired properties. □

The following statement immediately follows from the lemma shown above.

Theorem 5.1.2. *For both optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, the following holds: If the set of optimal schedules contains a schedule with exactly two jobs, it also contains a permutation schedule.*

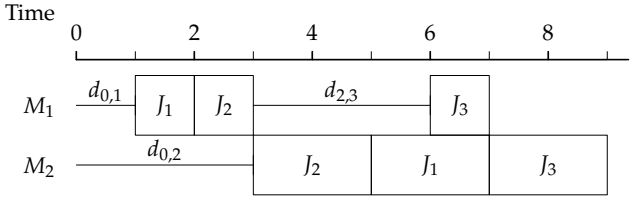
The value $n = 2$ is a sharp bound since for $n > 2$ instances can be constructed where the set of optimal schedules contains no permutation schedule.

Theorem 5.1.3. *For both optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, the following holds: For every $n > 2$ there exists an instance of the Two-Stage VRP with Profits and Buffers where the set of optimal schedules contains no permutation schedule.*

Proof. Consider the following instance with depot node v_0, 3 additional nodes v_1, v_2, v_3 and three corresponding jobs J_1, J_2, J_3. The processing times of the jobs are $a_J = 1$ for and $b_J = c = 2$ for all jobs J. The travel times d_{ij} are chosen according to the matrix

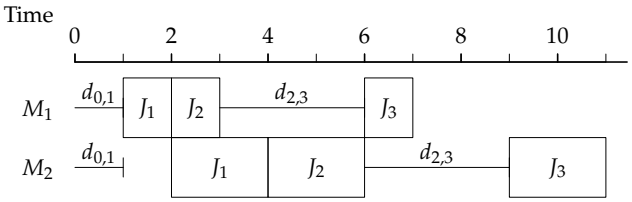
$$(d_{ij})_{\substack{i=0,1,2,3 \\ j=0,1,2,3}} = \begin{pmatrix} 0 & 1 & 3 & W \\ W & 0 & 0 & 0 \\ W & 0 & 0 & 3 \\ W & W & W & 0 \end{pmatrix},$$

where W can be $W = \infty$ or an appropriately large number. It is assumed that there are no buffer restrictions (or in other words, the buffer capacity Ω is chosen to be large enough that all jobs can be stored at the same time, with the consequence that the type of buffer and choice of s_J does not affect schedules). Next, consider the following non-permutation schedule σ^* with $\pi^1(\sigma^*) = (1, 2, 3)$ and $\pi^2(\sigma^*) = (2, 1, 3)$:

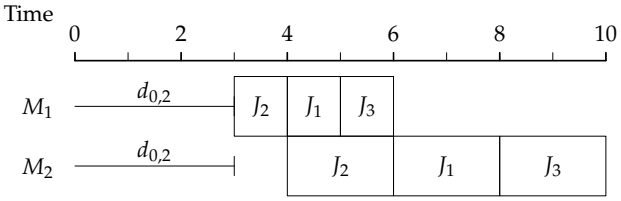


It can be shown that there exists no permutation schedule σ^P that reaches the same makespan as σ^* with $C_{\max}(\sigma^*) = 9$. This can be seen by testing all possible job permutations: Due to W and the choice of travel times d_{ij}, a schedule σ^P that

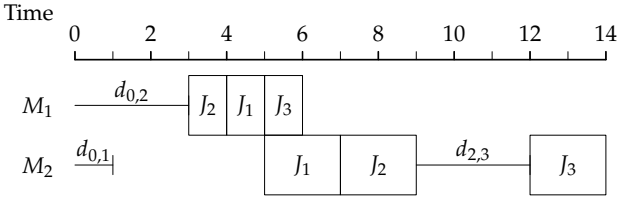
starts with J_3 takes at least W time units and cannot be shorter than σ^*. Next, assume that σ^P starts with J_1. The permutation $\pi = (J_1, J_3, J_2)$ can be excluded with the same reasoning, and the schedule σ_1 resulting from $\pi(\sigma_1) = (J_1, J_2, J_3)$ is also longer than σ^* with $C_{\max}(\sigma_1) = 11$:



Similarly with permutation schedules that start with J_2, the schedule with permutation $\pi(\sigma_2) = (J_2, J_3, J_1)$ has a length of at least W, and the permutation $\pi = (J_2, J_1, J_3)$ also leads to a schedule σ_2 longer than σ^* with $C_{\max}(\sigma_2) = 10$:



In short, only job permutations $\pi^m = (J_1, J_2, J_3)$ and $\pi^m = (J_2, J_1, J_3)$ (with $m \in \{1, 2\}$) lead to schedules that can be potentially shorter than W. This can be used to show that σ^* is a makespan-minimizing schedule as the remaining cases to check can be restricted to these permutations. In particular, all combinations except $\pi^1 = (J_2, J_1, J_3)$ and $\pi^2 = (J_1, J_2, J_3)$ are already dealt with above, so it is only remains to check this case:



This non-permutation schedule σ_4 is also longer than σ^* with $C_{\max}(\sigma_4) = 14$. Since all other cases were shown to lead to longer schedules, it follows that σ^* is an optimal non-permutation schedule and that there exists no permutation schedule with the same length.

The set of optimal schedules can be appropriately restricted to only contain σ^* by choosing Q to be large enough so that all jobs need to be processed (in the case of $\mathcal{O}_{\min C_{\max},\, R \geq Q}$)) or by choosing the budget B as $B = 9$ so that σ^* is the only schedule that maximizes the profit (by processing all jobs) without violating the budget constraint.

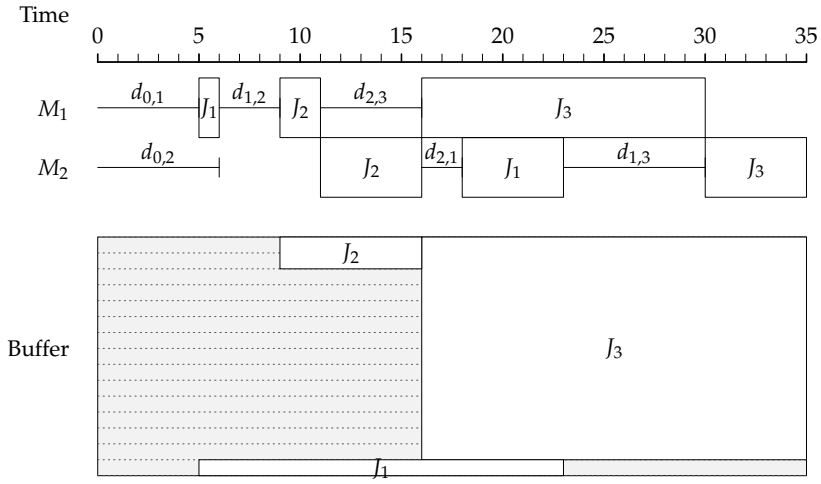
The presented instance can be extended to contain more than 3 jobs by adding "extra nodes" v_ℓ with "extra jobs" J_ℓ and processing times $a_{J_\ell} = c$ and appropriately choosing the travel times $d_{i,\ell}$ and $d_{\ell,i}$ such that the extra jobs have to be processed at the end in succession after J_3, or else travel times of length W are induced. □

The instance presented in the proof above relies on large numbers W in order to be applicable regardless of buffer type (intermediate buffer or spanning buffer) and buffer usage ($s_J = a_J$ or $s_J = 1$ for all J). However, it is also possible to construct instances where the property that no optimal permutation schedules exist arises due to the buffer. For the sake of completeness, two examples for such instances are given in the following including examples for optimal non-permutation schedules. However, the proofs that the set of optimal schedules for these instances contains no permutation schedules are omitted as they rely on the same method used in the proof above, namely, enumerating and checking all permutation schedules (which is straightforward to do by hand).

The first example contains a depot node v_0 and three additional nodes with jobs J_1, J_2, J_3 that have processing times $a_{J_1} = 1, a_{J_2} = 2, a_{J_3} = 14$ and $c = 5$. In addition, a spanning buffer is used with $\Omega = 15$ and $s_J = a_J$. The travel times are chosen according to the following matrix:

$$(d_{ij})_{\substack{i=0,1,2,3 \\ j=0,1,2,3}} = \begin{pmatrix} 0 & 5 & 5 & 5 \\ 5 & 0 & 2 & 3 \\ 6 & 3 & 0 & 2 \\ 4 & 7 & 5 & 0 \end{pmatrix}$$

One example for an optimal non-permutation schedule σ^* uses the permutations $\pi^1(\sigma^*) = (J_1, J_2, J_3)$ and $\pi^2(\sigma^*) = (J_2, J_1, J_3)$:



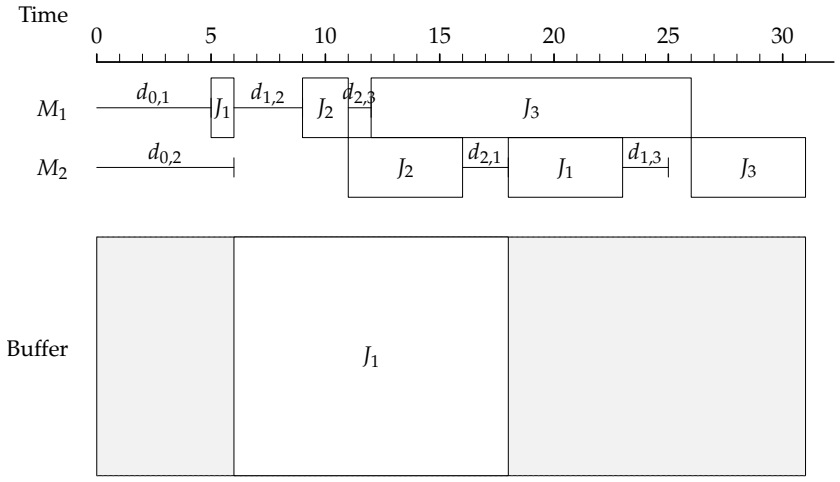
This instance can be extended by adding "extra nodes" v_ℓ and jobs J_ℓ with $a_{J_\ell} = \Omega$ and $d_{\ell,3} = d_{3,\ell} = 0$, $d_{\ell,j} = d_{3,j}$, $d_{j,\ell} = d_{j,3}$ for all $j \notin \{3, \ell\}$ as well as $d_{\ell,\ell} = 0$ and $d_{\ell,\ell'} = d_{\ell',\ell} = 0$ for all extra nodes $v_{\ell'} \neq v_\ell$. With these values, an optimal schedule needs to process all extra jobs in succession at the end of the schedule or else travel times are induced.

For the resulting instances it can be shown similar to the proof above that the shortest non-permutation schedule σ^* that processes all jobs has a length $C_{\max}(\sigma^*)$ that cannot be reached by any permutation schedule. By appropriately choosing the budget constraint or minimum score constraint, it is possible to obtain instances for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ and $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ in which the set of optimal schedules contains no permutation schedules.

The second example also uses three nodes (excluding the depot node v_0) with jobs J_1, J_2, J_3 that have processing times $a_{J_1} = 1, a_{J_2} = 2, a_{J_3} = 14$ and $c = 5$. However, an intermediate buffer is used with $\Omega = 1$ and $s_J = a_J$. The travel times are chosen as follows:

$$(d_{ij})_{\substack{i=0,1,2,3 \\ j=0,1,2,3}} = \begin{pmatrix} 0 & 5 & 5 & 5 \\ 5 & 0 & 2 & 3 \\ 6 & 3 & 0 & 2 \\ 4 & 2 & 1 & 0 \end{pmatrix}$$

One of the optimal non-permutation schedules σ^* for this instance has the permutations $\pi^1(\sigma^*) = (J_1, J_2, J_3)$ and $\pi^2(\sigma^*) = (J_2, J_1, J_3)$:



This instance can be extended to $n > 3$ by adding nodes v_ℓ and jobs J_ℓ with processing times $a_{J_\ell} = c$. The travel lengths $d_{\ell,j}$ and $d_{j,\ell}$ are set in the same way as in the previous example.

5.1.3 The Gap Between Permutation Schedules an Non-Permutation Schedules

Since it was shown in the previous section that there exist instances of the Two-Stage VRP with Profits and Buffers where the set of optimal schedules contains no permutation schedule, the question arises as to how large the "gap" in solution quality between permutation schedules and non-permutation schedules can potentially become. In the following, some statements are presented that deal with this question.

The Optimization Problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$

First, the optimization problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ is considered where the makespan $C_{\max}(\sigma)$ needs to be minimized while ensuring that a minimum profit Q is collected, i.e., $R(\sigma) \geq Q$. In the following, let $C_{\max}(\sigma^*, \mathcal{I})$ be the makespan of an optimal schedule for an instance $\mathcal{I}$ and let $C_{\max}(\sigma^*_{perm}, \mathcal{I})$ be the makespan of a "best" permutation schedule (i.e., a permutation schedule that minimizes the makespan over all permutation schedules). Since the Two-Stage VRP with Profits and Buffers encompasses the Two-Machine Flow Shop with Buffers, the examples given in

Theorem 3.2.7 and Theorem 3.2.6 for Two-Machine Flow Shops with buffers (see Section 3.2) also apply to the more general problem by choosing the required profit Q such that all jobs need to be processed.

Corollary 5.1.1. *Assume that $s_J = a_J$ holds for all jobs J. For the buffer type* bufType $=$ *intermediateBuffer, there exists a sequence of instances $\mathcal{I}_k$ with $k = 1, 2, 3, \ldots$ with*

$$\lim_{k\to\infty} \frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k} = \frac{7}{6}.$$

For bufType $=$ *spanningBuffer, there exists a sequence of instances $\mathcal{I}_k$ with $k = 1, 2, 3, \ldots$ with*

$$\lim_{k\to\infty} \frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k} = \frac{19}{18}.$$

However, the above statement only refers to a sequence of instances and the asymptotical makespan ratio between the best permutation schedule and an optimal schedule. Another example that uses a specific instance was given in the proof of Theorem 5.1.3 which has the following makespan ratio.

Corollary 5.1.2. *For* bufType $\in \{$*intermediateBuffer*, *spanningBuffer*$\}$ *and* bufUsage $\in \{$"$s_J = 1$", "$s_J = a_J$"$\}$, *there exists an instance $\mathcal{I}$ with*

$$\frac{C_{\max}(\sigma^*_{perm}, \mathcal{I})}{C_{\max}(\sigma^*, \mathcal{I})} = \frac{10}{9}.$$

The statements presented above show examples on how large the "gaps" in solution quality can become. However, it can be shown that this ratio cannot be larger than 2 for $\mathcal{O}_{\min C_{\max},\, R\geq Q}$. In order to see this, consider the following lemma.

Lemma 5.1.3. *Let σ be a non-permutation schedule for an instance of the Two-Stage VRP with Profits and Buffers. Then, there exists a permutation schedule σ^P with the properties $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$ and $R(\sigma^P) = R(\sigma)$ that satisfies $C_{\max}(\sigma^P) \leq 2 \cdot C_{\max}(\sigma)$.*

Proof. Assume that σ is a non-permutation schedule, and let $ProcessedJobs(\sigma)$ be the set of jobs processed in σ with $s = |ProcessedJobs|$. Let D_1, D_2 be the sum of travel times d_{ij} taken on the paths $P^1(\sigma), P^2(\sigma)$ (i.e., paths of nodes traversed in the graph) by M_1 and M_2, respectively. A trivial lower bound for the makespan of σ

is $C_{\max}(\sigma) \geq \max\{\sum a_J + D_1,\ s \cdot c + D_2\}$, where the sum is taken over all jobs in $\mathit{ProcessedJobs}(\sigma)$.

Next, construct the following permutation schedule σ^P. For $k = \arg\min_{k \in \{1,2\}} D_k$ both vehicles traverse the nodes in the order $P^k(\sigma)$ and process the corresponding jobs in the order $\pi(\sigma^P) = \pi^k(\sigma)$. Without loss of generality, it is assumed that $\pi(\sigma^P) = (J_1, J_2, \ldots, J_s)$. The processing of jobs in σ^P is done such that the buffer is used as little as possible. Formally, this means that

$$
\begin{aligned}
S_1^1(\sigma^P) &= 0 \\
S_1^2(\sigma^P) &= C_1^1(\sigma^P) \\
S_\ell^1(\sigma^P) &= \max\{C_{\ell-1}^2(\sigma^P),\ C_{\ell-1}^1(\sigma^P) + d_{\ell-1,\ell}\} \\
S_\ell^2(\sigma^P) &= \max\{C_\ell^1(\sigma^P),\ C_{\ell-1}^2(\sigma^P) + d_{\ell-1,\ell}\}
\end{aligned}
$$

for $\ell = 2, 3, \ldots, s$.

In the intermediate buffer model, the buffer is not used, whereas for the spanning buffer not more than $\max_{J \in \mathit{ProcessedJobs}(\sigma)} s_J$ buffer units are used which is a lower bound for the buffer units used in the original schedule σ (or else σ would violate the buffer constraint). Thus, σ^P is feasible with respect to buffer usage. In addition, this schedule satisfies $\mathit{ProcessedJobs}(\sigma^P) = \mathit{ProcessedJobs}(\sigma)$ and $R(\sigma^P) = R(\sigma)$.

To analyze the makespan $C_{\max}(\sigma^P)$ of σ^P, note that at any point in time t, exactly one of the following three cases hold due to the definition of σ^P:

1. $t \in T_A$: M_1 is processing a job.
2. $t \in T_B$: M_2 is processing a job.
3. $t \in T_C$: None of the machines are processing a job.

Note that for $t \in T_C$ it is not possible that both machines are idle so that in the case $t \in T_C$ at least one of the machines must be traversing an edge e_{ij} (with travel times d_{ij}).

Using this notation, it is possible to give upper bounds on the number of time units contained in T_A, T_B and T_C. Trivial upper bounds for the former two are $T_A \leq \sum a_J$ (with the sum being taken over all jobs $J \in \mathit{ProcessedJobs}(\sigma^P)$) and $T_B \leq s \cdot c$.

Regarding T_C, note that it is possible that one machine traverses an edge with the other machine waiting for the next job (and vice versa), so it is not sufficient to only

consider the edge travel times for one of the machines M_1, M_2. However, adding the factor 2 allows for the following upper bound that fully contains the edge travel times for both vehicles:

$$T_C \leq 2 \cdot \min\{D_1, D_2\}$$

These upper bounds can be used to calculate an upper bound for $C_{\max}(\sigma^P)$:

$$\begin{aligned} C_{\max}(\sigma^P) &\leq \sum a_J + s \cdot c + 2\min\{D_1,\ D_2\} \\ &\leq \sum a_J + D_1 + s \cdot c + D_2 \\ &\leq 2 \cdot \max\{\sum a_J + D_1,\ s \cdot c + D_2\} \\ &\leq 2 \cdot C_{\max}(\sigma) \end{aligned}$$

Thus, σ^P is the permutation schedule with the desired properties. □

A "best" permutation schedule σ^*_{perm} (with respect to $\mathcal{O}_{\min C_{\max},\, R \geq Q}$) cannot be worse than the permutation schedule σ^P constructed in the proof of the previous lemma so that the following statement holds.

Theorem 5.1.4. *For any instance of the Two-Stage VRP with Profits and Buffers with* bufType $\in \{$*intermediateBuffer*, *spanningBuffer*$\}$*, the following upper bound holds for the solution quality ratio* $C_{\max}(\sigma^*_{perm}, \mathcal{I}) / C_{\max}(\sigma^*, \mathcal{I})$ *between a best permutation schedule* σ^*_{perm} *and an optimal schedule* σ^**:*

$$\frac{C_{\max}(\sigma^*_{perm}, \mathcal{I}_k)}{C_{\max}(\sigma^*, \mathcal{I}_k)} \leq 2$$

The Optimization Problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$

Next, the optimization problem is considered where the collected profit $R(\sigma)$ is to be maximized without violating the budget constraint $C_{\max}(\sigma) \leq B$. In the following, let $R(\sigma^*, \mathcal{I})$ be the total value collected in an optimal schedule for an instance $\mathcal{I}$ and let $R(\sigma^*_{perm}, \mathcal{I})$ be the total profit collected in a "best" permutation schedule (i.e., a permutation schedule that maximizes the total profit over all permutation schedules that do not violate the budget constraint).

First, a straightforward example can be given where $R(\sigma^*, \mathcal{I}) / R(\sigma^*_{perm}, \mathcal{I}) = 3/2$ holds. Recall the example instance $\mathcal{I}$ with 3 jobs given in the proof of Theorem 5.1.3

where there existed a non-permutation schedule σ^* with makespan $C_{\max}(\sigma^*) = 9$ and all permutation schedules with 3 jobs had a makespan of 10 or more time units. By choosing the budget B as $B = 9$, the set of permutation schedules that do not violate the budget constraint can only contain schedules where at most 2 jobs are processed.

Without loss of generality, it is assumed that r_1, r_2, r_3 are the profit values of the three jobs processed in the non-permutation schedule σ^* with $r_1 \leq r_2 \leq r_3$ and it is assumed that the best permutation schedule σ^*_{perm} has the total profit $R(\sigma^*_{perm}) = r_2 + r_3$ (i.e., it processes the two most valuable jobs out of the three). In addition, let $R = r_1 + r_2 + r_3$ be the total profit of the three jobs.

Then, the largest ratio in solution quality between σ^* and σ^*_{perm} is reached when the profits are chosen as $r_1 = r_2 = r_3 = R/3$. In order to see this, consider the optimization problem of maximizing the ratio $\frac{r_1+r_2+r_3}{r_2+r_3}$ and substitute r_1 using $r_1 = R - r_2 - r_3$. This leads to the ratio $\frac{R}{r_2+r_3}$ which is maximized if and only if $r_2 + r_3$ is minimized. Note that the function $g(r_1, r_2, r_3) = r_2 + r_3$ is a linear function so that the aforementioned values $r_1 = r_2 = r_3 = R/3$ can be obtained in a straightforward manner by applying the simplex algorithm to the linear program that results from the constraints $r_1 \leq r_2 \leq r_3$ and $r_1 + r_2 + r_3 = R$, where the objective function $g(r_1, r_2, r_3) = r_2 + r_3$ is to be minimized.

Using the values $r_1 = r_2 = r_3 = R/3$, the resulting profits are $R(\sigma^*) = R$ and $R(\sigma^*_{perm}) = (2/3)R$ which gives an example for the following ratio.

Theorem 5.1.5. *There exists an instance $\mathcal{I}$ of the Two-Stage VRP with Profits and Buffers with* bufType $\in \{$*intermediateBuffer*, *spanningBuffer*$\}$ *with the solution quality ratio $R(\sigma^*, \mathcal{I})/R(\sigma^*_{perm}, \mathcal{I}) = 3/2$ for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$.*

Regarding upper bounds for the ratio, note that the problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ is different from $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ in that extending the length $C_{\max}(\sigma)$ of a given solution σ can lead to σ violating the budget constraint $C_{\max}(\sigma) \leq B$. However, it is possible to modify a given solution σ to a new solution σ' in a way such that the same nodes v are visited, but not all of their corresponding jobs J_v are processed. This decreases the quality of the solution, but makes it possible to construct certain permutation schedules that are guaranteed to not violate the budget constraint.

Lemma 5.1.4. *Let σ^* be a non-permutation schedule for an instance of the Two-Stage VRP with Profits and Buffers where the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ is considered.*

Assume that $\mathit{ProcessedJobs}(\sigma^*) > 2$. *Then, there exists a permutation schedule* σ^P *that satisfies* $C_{\max}(\sigma^P) \leq C_{\max}(\sigma^*)$ *while processing the two jobs with the highest value in* $\mathit{ProcessedJobs}(\sigma^*)$

Proof. Without loss of generality, assume that $\mathit{ProcessedJobs}(\sigma^*) = \{J_1, J_2, \ldots, J_\ell\}$ with the profits $r_{J_1} \leq r_{J_2} \leq \cdots \leq r_{J_\ell}$ (but it is not required that the jobs are processed in that order) and $\ell > 2$. Next, consider a "truncated" version $\tilde{\sigma}$ of σ^* where the machines M_1, M_2 traverse the same path as in σ^* (i.e., $P^1(\tilde{\sigma}) = P^1(\sigma^*)$ and $P^2(\tilde{\sigma}) = P^2(\sigma^*)$), but only processes the two most valuable jobs J_ℓ and $J_{\ell-1}$, whereas for all other jobs in $\mathit{ProcessedJobs}(\sigma^*)$ the machines wait at their respective nodes so that the starting and completion times for J_ℓ and $J_{\ell-1}$ as well as $C_{\max}(\tilde{\sigma})$ are identical to the times in σ^*.

Since in the non-permutation schedule $\tilde{\sigma}$ exactly 2 jobs are processed, it is possible to apply Lemma 5.1.2 which states that non-permutation schedules where two jobs are processed can be reordered to permutation schedules without an increase in length or a decrease in collected profit. Applying this lemma to $\tilde{\sigma}$ leads to a permutation schedule σ^P with the desired properties. □

The shown lemma guarantees the existence of permutation schedules σ^P that collect the profits of the two most valuable jobs from a given non-permutation schedule σ^*. This raises the question as to how the profits $r_{J_1}, r_{J_2}, \ldots, r_{J_\ell}$ should be chosen so that the solution quality ratio between σ^* and σ^P becomes as large as possible. It can be shown that, similar to the example above with 3 jobs, the largest ratio is obtained when the profit is equally distributed over all jobs $J_1, J_2, \ldots, J_\ell$.

Lemma 5.1.5. *Assume that a non-permutation schedule* σ^* *processes* $\ell > 2$ *jobs and collects the total profit* $R(\sigma^*) = R$. *Assume that a permutation schedule* σ^P *processes the two jobs with the highest value in* $\mathit{ProcessedJobs}(\sigma^*)$. *Then, the ratio* $R(\sigma^*)/R(\sigma^P)$ *is maximized when all jobs* $J \in \mathit{ProcessedJobs}(\sigma^*)$ *have the profit* $r_J = R/\ell$.

Proof. Without loss of generality, let $\mathit{ProcessedJobs}(\sigma^*) = \{J_1, J_2, \ldots, J_\ell\}$ and let $r_1, r_2, \ldots, r_\ell$ be the profit values for these jobs with $r_1 \leq r_2 \leq \cdots \leq r_\ell$ so that σ^P processes the jobs J_ℓ and $J_{\ell-1}$.

First, it can be shown, that the problem of maximizing the ratio entails the solution of a linear optimization problem, similar to the example presented above with 3 jobs.

In particular, note that $r_1 + r_2 + \cdots + r_\ell = R$ so that the term to be maximized can be expressed as

$$\frac{R(\sigma^*)}{R(\sigma^P)} = \frac{r_1 + r_2 + \cdots + r_{\ell-1} + r_\ell}{r_{\ell-1} + r_\ell} = \frac{R}{r_{\ell-1} + r_\ell}.$$

Since the total profit R is not a variable, it follows that this term is maximized when $r_{\ell-1} + r_\ell$ is minimized (or equivalently, if $-(r_{\ell-1} + r_\ell)$ is maximized, which is required further below). For this reason, it is sufficient to consider the following linear optimization problem that is obtained from the inequalities $r_1 \leq r_2 \leq \cdots \leq r_\ell$ and the constraint $r_1 + r_2 + \cdots + r_\ell = R$:

$$\text{Maximize} \quad f(r_1, r_2, \ldots, r_{\ell-1}, r_\ell) = -r_{\ell-1} - r_\ell$$

subject to $r_1, r_2, \ldots, r_{\ell-1}, r_\ell \geq 0$ and

$$\begin{aligned}
&(y_i) && r_i - r_{i+1} && \leq 0 && (i = 1, 2, \ldots, \ell - 1)\\
&(y_s) && r_1 + r_2 + \cdots + r_{\ell-1} + r_\ell && = R, &&
\end{aligned}$$

where the terms "(y)" are used as names for the constraints. This optimization problem contains ℓ variables, $\ell - 1$ inequality constraints and one equality constraint.

Since the number of constraints is not fixed, the *duality theorems for linear programming* (see, e.g., [25, 53]) are used to construct a solution $\underline{r} = (r_1, r_2, \ldots, r_{\ell-1}, r_\ell)$ for this linear program and establish its optimality. In particular, the weak duality theorem is a statement about a given linear program (the "primal linear program") where an objective function f is to be *maximized*, and its dual linear program with the dual objective function g that is to be minimized. It states that the objective function values $g(\underline{y})$ for all feasible values $\underline{y}$ of the dual program are upper bounds for the objective function f of the primal linear program. Thus, if a solution $\underline{r}^*$ for the primal program and a solution $\underline{y}^*$ for the dual program are found with the property $f(\underline{r}^*) = g(\underline{y}^*)$, it follows that $\underline{r}^*$ and $\underline{y}^*$ are optimal solutions for their respective linear programs. That this property holds for all linear programs, where the primal and dual linear programs have feasible solutions, is known as the strong duality theorem [25].

In short, in order to show that a choice for the values $\underline{r} = (r_1, r_2, \ldots, r_{\ell-1}, r_\ell)$ obtains the maximum with respect to objective function f, it suffices to present a feasible solution $\underline{y}$ for the dual program that obtains the same value. For this, the dual program needs to be constructed first by applying the construction rules for dual linear programs (which are described, e.g., in [25]). Note that constraints y and variables r of the primal linear program become variables and constraints in the dual program, respectively:

$$\text{Minimize} \quad g(y_1, y_2, \ldots, y_{\ell-1}, y_s) = R \cdot y_s$$

subject to $y_1, y_2, \ldots, y_{\ell-1} \geq 0$, $y_s \in \mathbb{R}$ and

$$\begin{aligned}
(r_1) \quad & y_1 + y_s && \geq 0 \\
(r_2) \quad & -y_1 + y_2 + y_s && \geq 0 \\
(r_3) \quad & -y_2 + y_3 + y_s && \geq 0 \\
& \vdots \\
(r_{\ell-2}) \quad & -y_{\ell-3} + y_{\ell-2} + y_s && \geq 0 \\
(r_{\ell-1}) \quad & -y_{\ell-2} + y_{\ell-1} + y_s && \geq -1 \\
(r_\ell) \quad & -y_{\ell-1} + y_s && \geq -1,
\end{aligned}$$

where the terms "(r)" are the "names" of constraints in the dual linear program. This linear program has ℓ variables $y_1, y_2, \ldots, y_{\ell-1}, y_s$ and ℓ constraints..

Next, consider the solution $\underline{r}^*$ for the primal linear program where all variables r_i have the value $r_i = R/\ell$ (for $i \in \{1, 2, \ldots, \ell\}$). This solution obtains the objective value $f(\underline{r}^*) = -2R/\ell$ and it is straightforward to check that this solution satisfies all constraints.

For the dual linear program, consider the solution $\underline{y}^* = (y_1, y_2, \ldots, y_{\ell-2}, y_{\ell-1}, y_s)$ with the following values:

$$\begin{aligned}
y_i &= (2/\ell) \cdot i && \text{(for } i \in \{1, 2, \ldots, \ell - 2\}\text{)} \\
y_{\ell-1} &= (2/\ell) \cdot \left(\frac{\ell}{2} - 1\right) \\
y_s &= (2/\ell) \cdot (-1)
\end{aligned}$$

It can be immediately seen that $g(\underline{y}^*) = f(\underline{r}^*) = -2R/\ell$ so that it only remains to check whether $\underline{y}^*$ satisfies all constraints in order to show that $\underline{r}^*$ is an optimal solution. It is straightforward to see that constraint (r_1) is satisfied: $y_1 + y_s = (2/\ell) \cdot [1 - 1] = 0$. For constraints (r_k) with $k \in \{2, 3, \ldots, \ell - 2\}$, observe that $-y_{k-1} + y_k + y_s = (2/\ell) \cdot [-(k-1) + k - 1] = (2/\ell) \cdot 0 = 0$, which means that these constraints are not violated. Regarding constraint $(r_{\ell-1})$, note that $y_{\ell-2} + y_{\ell-1} + y_s = (2/\ell) \cdot [-(\ell - 2) + (\frac{\ell}{2} - 1) - 1] = (2/\ell) \cdot [-\frac{\ell}{2}] = -1$ so that this constraint is also satisfied. Finally, constraint (r_ℓ) holds since $-y_{\ell-1} + y_s = (2/\ell) \cdot [-\frac{\ell}{2} + 1 - 1] = -1$.

Thus, $\underline{y}^*$ is a feasible solution for the dual linear program. Since it obtains the same objective value as $\underline{r}^*$ in the primal linear program, it follows from the strong duality theorem that $\underline{r}^*$ is an optimal solution for the primal linear program that maximizes the ratio $R/(r_{\ell-1} + r_\ell)$.

□

It is interesting to note that in the proof above the ratio $R(\sigma^*)/R(\sigma^P)$ can be expressed as $\frac{R}{r_{\ell-1}+r_\ell}$ which might indicate that the value of the ratio does not directly depend on the smaller profit values $r_1, r_2, \ldots, r_{\ell-2}$, but only on $r_{\ell-1}, r_\ell$ and R. In fact, setting $r_\ell = R/\ell$ for the highest profit value determines the remaining profits as it inductively follows from the constraints $r_1 \leq r_2 \leq \cdots \leq r_\ell$ and $r_1 + r_2 + \cdots + r_\ell = R$ that the other r_i must also have the value $r_i = R/\ell$.

In the case where the profit values are chosen as described in Lemma 5.1.5, the permutation schedule σ^P has the total profit $R(\sigma^P) = 2R/\ell$ from which one obtains the ratio $R(\sigma^*)/R(\sigma^P) = \ell/2$, where ℓ is the number of jobs processed in the non-permutation schedule σ. Since it is possible that $\ell = n$, which means that all n jobs are processed in σ, an upper bound for the ratio $R(\sigma^*)/R(\sigma^P)$ is as follows.

Theorem 5.1.6. *For an instance $\mathcal{I}$ of the Two-Stage VRP with Profits and Buffers containing n jobs with* bufType $\in \{$*intermediateBuffer*, *spanningBuffer*$\}$*, the solution quality ratio $R(\sigma^*, \mathcal{I})/R(\sigma^*_{perm}, \mathcal{I})$ for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ satisfies*

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}\, \mathcal{I})} \leq \frac{n}{2}.$$

Since this upper bound (for the general case of the Two-Stage VRP with Profits and Buffers) depends on the number of jobs n that can be processed in a given instance,

it might be interesting to investigate whether tighter bounds can be established for restricted cases of the Two-Stage VRP with Profits and Buffers.

5.1.4 Remarks on Restricted Cases

In this section, special cases of the Two-Stage VRP with Profits and Buffers are analyzed that contain restrictions similar to the Two-Machine Flow Shop with Buffers and the Orienteering Problem, such as having zero processing times, having processing times on one machine that dominate the processing times on the other machine, or no travel times on the edges. A high-level overview of the restrictions and special cases considered in this section, including the obtained results is later given in Section 5.1.5.

Computational Complexity

The next statement is similar to Theorem 4.2.2 which states that the Orienteering Problem is *NP*-complete. This lemma considers the Two-Stage VRP with Profits and Buffers with the same restrictions (i.e., no processing times on any of the machines, no buffer restrictions, but non-zero travel times d_{ij} on the edges are allowed), but states that the problem of minimizing the makespan is still *NP*-complete even when the sub-problem of selecting a suitable subset of jobs is removed by requiring that all jobs need to be processed.

Theorem 5.1.7. *For a given instance of the Two-Stage VRP with Profits and Buffers without buffer constraints, but the additional restriction that all processing times on both machines are zero, the decision problem whether for a given number B there exists a feasible schedule σ that processes all jobs and satisfies $C_{\max}(\sigma) \leq B$ is NP-complete.*

Proof. This restricted case of the Two-Stage VRP with Profits and Buffers is in *NP* since it can be verified in polynomial time whether a schedule σ satisfies $C_{\max}(\sigma) \leq B$. Regarding *NP*-hardness, it is not hard to see that this problem is very similar to the decision variant of the Traveling Salesperson Problem (TSP), which asks whether in a directed graph with edge weights a Hamiltonian path[1] exists where

[1] Another common formulation of the TSP uses Hamiltonian cycles where the start node and end node needs to be equal. However, removing this condition and permitting paths which are not cycles does not change the complexity of the TSP [148].

the sum of traversed edge weights (the "total length" of that path) does not exceed a value L. This decision problem is known to be *NP*-complete [59].

A reduction from a TSP instance $\mathcal{I}_{TSP}$ to an instance $\mathcal{I}_{VRP'}$ of the restricted case is straightforward to do by using the same graphs with the same edge weights, setting all jobs on all nodes to have zero processing times and setting $B = L$. It is not hard to see that if a solution for $\mathcal{I}_{TSP}$ exists where the total length does not exceed L, then a schedule σ exists that satisfies $C_{\max}(\sigma) \leq B$. For the inverse direction, note that since the processing times on both machines are all zero, any non-permutation schedule σ can be converted to a permutation schedule σ^P without increasing $C_{\max}$ (for example, by setting $\pi(\sigma^P) = \pi^2(\sigma)$ and $S_J^1(\sigma^P) = S_J^2(\sigma^P) = S_J^2(\sigma)$ for all $J \in \mathit{ProcessedJobs}(\sigma)$ so that M_1 finishes the last job at the same time as M_2). The sequence of traversed nodes in σ^P can then be used to construct a path for $\mathcal{I}_{TSP}$ where the total length does not exceed L. □

The following theorem and its proof shows that another restricted variant of the Two-Stage VRP with Profits and Buffers also encompasses aspects of the Knapsack Problem, another well-known *NP*-hard problem, since the latter problem can be reduced to the former problem.

Theorem 5.1.8. *For a given instance of the Two-Stage VRP with Profits and Buffers without buffer constraints, without travel times d_{ij} on edges, but with the additional restriction that $c = 0$, the decision problem whether for two given numbers $B \geq 0$, $Q \geq 0$ there exists a feasible schedule σ that satisfies $C_{\max}(\sigma) \leq B$ and $R(\sigma) \geq Q$ is NP-complete.*

Proof. This restricted version of the Two-Stage VRP with Profits and Buffers is in *NP* since it can be verified in polynomial time whether a solution σ satisfies $C_{\max}(\sigma) \leq B$ and $R(\sigma) \geq Q$. In order to show *NP*-hardness, a reduction from the Knapsack problem is used. The decision variant (which is known to be *NP*-complete [55]) asks whether for a set of n items $x_1, x_2, \ldots, x_n$ that each have weights $w(x_i) \geq 0$ and scores $s(x_i) \geq 0$ ($i \in \{1, 2, \ldots, n\}$), and two given numbers W, Y there exists a subset $\mathcal{S}$ of these items such that $\sum_{x \in \mathcal{S}} w(x_i) \leq W$ and $\sum_{x \in \mathcal{S}} \geq Y$.

Given an instance $\mathcal{I}_{KP}$ of the Knapsack problem, an instance $\mathcal{I}_{VRP'}$ of the restricted VRP case can be constructed using a complete graph with n nodes $v_1, v_2, \ldots, v_n$ and n corresponding jobs $J_1, J_2, \ldots, J_n$ (that correspond to the n items x_i) where the profit values are $r_{J_i} = s(x_i)$ and the processing times on M_1 are $a_{J_i} = w(x_i)$ for

$i \in \{1, 2, \ldots, n\}$. In addition, the M_2 processing times are $b_{J_i} = c = 0$ for all jobs and all edges e_{ij} have travel time $d_{ij} = 0$. Furthermore, the values B and Q for the budget constraint and minimum score constraint, respectively, are chosen as $B = W$ and $Q = Y$.

It is not hard to see that for a solution σ of $\mathcal{I}_{VRP'}$ (where all jobs are processed as early as possible), its makespan $C_{\max}(\sigma)$ and total profit $R(\sigma)$ only depends on $\mathit{ProcessedJobs}(\sigma)$ and not the order in which the jobs are processed. For this reason, it is straightforward to transform a solution $\mathcal{S}$ for $\mathcal{I}_{KP}$ that satisfies $\sum_{x \in \mathcal{S}} w(x_i) \leq W$ and $\sum_{x \in \mathcal{S}} \geq Y$ into a solution σ of $\mathcal{I}_{VRP'}$ that satisfies $C_{\max}(\sigma) \leq B$ and $R(\sigma) \geq Q$ (by constructing a permutation schedule where all jobs are processed as early as possible in any order).

For the inverse direction, assume that a schedule σ exists with $C_{\max}(\sigma) \leq B$ and $R(\sigma) \geq Q$. A lower bound for $C_{\max}(\sigma)$ is $C_{\max}(\sigma) \geq \sum_{J_i} a_J$ with $J_i \in \mathit{ProcessedJobs}(\sigma)$. Since it is assumed that $C_{\max}(\sigma) \leq B$, it follows that $\sum_{J_i} a_{J_i} \leq B$ where the sum is taken over $J \in \mathit{ProcessedJobs}(\sigma)$. Regarding $\mathcal{I}_{KP}$, it is straightforward to construct a subset $\mathcal{S}$ based on $\mathit{ProcessedJobs}(\sigma)$ (by choosing the items x_i corresponding to the jobs J_i). Due to $\sum_{J_i} a_{J_i} \leq B$ and $B = W$, subset $\mathcal{S}$ satisfies $\sum_{x \in \mathcal{S}} w(x_i) = \sum_{J_i} a_{J_i} \leq W$. Regarding scores, $\mathcal{S}$ also satisfies $\sum_{x \in \mathcal{S}} \geq Y$ since $R(\sigma) \geq Q$ and $Q = Y$. This shows that $\mathcal{S}$ is a solution for $\mathcal{I}_{KP}$.

□

When the Two-Stage VRP with Profits and Buffers is further restricted, a special case can be obtained where the same decision problem is efficiently solvable. In fact, the two optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ corresponding to that decision problem can be efficiently solved from which it directly follows that the decision problem is in P.

Theorem 5.1.9. *Consider an instance of the Two-Stage VRP with Profits and Buffers with* bufType $\in \{\mathit{intermediateBuffer}, \mathit{spanningBuffer}\}$ *and* bufUsage $\in \{s_J = a_J, s_J = 1\}$ *where* $a_J = 0$ *holds for all jobs* J *and* $d_{ij} = 0$ *holds for all edges* e_{ij} *in the graph. Then, for both optimization problems* $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ *and* $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ *a feasible permutation schedule that is optimal can be constructed in polynomial time.*

Proof. Due to the restrictions $a_J = 0$ for all jobs J and d_{ij} for all edges e_{ij}, for any subset of jobs $\mathcal{J}_1 \subseteq \mathcal{J}$ it is possible to construct a permutation schedule σ^P that

satisfies $C_{\max}(\sigma^P) = c \cdot |\mathcal{J}_1|$ (by processing all jobs in $\mathcal{J}_1$ as early as possible in any order). This is a trivial lower bound for any schedule σ with $\mathit{ProcessedJobs}(\sigma) = \mathcal{J}_1$, so it is not possible for non-permutation schedules to have a shorter makespan than $c \cdot |\mathcal{J}_1|$.

For the optimization problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ (minimizing makespan while obtaining a minimum profit Q), an optimal solution can be constructed by starting with $\mathcal{J}_1 = \varnothing$ and repeatedly adding the job with highest profit to $\mathcal{J}_1$ until the minimum score constraint is satisfied, i.e., $\sum_{J \in \mathcal{J}_1} r_J \geq Q$. Constructing a permutation schedule σ^P as described above leads to a schedule that minimizes the makespan $C_{\max}(\sigma^P) = c \cdot |\mathcal{J}_1|$ since the number of jobs in $\mathcal{J}_1$ is minimal.

For the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ (maximize profit without exceeding the budget B), an optimal solution can be constructed as follows. Note that $\ell_1 = \lfloor B/c \rfloor$ is the highest number of jobs that can be in a schedule σ without violating the budget constraint, since for any schedule σ that processes ℓ jobs a lower bound for the makespan is $C_{\max}(\sigma) \geq \ell \cdot c$. Choosing the ℓ_1 jobs with the highest profit and constructing a permutation schedule σ^P as described above using these jobs leads to a solution that satisfies $C_{\max}(\sigma^P) = \ell_1 \cdot c \leq B$ (the lower bound for any schedule with ℓ_1 jobs) while maximizing the total profit $R(\sigma^P)$, since any change to $\mathit{ProcessedJobs}(\sigma^P)$ would lead to a new solution σ' that violates the budget constraint or has a profit $R(\sigma') \leq R(\sigma^P)$.

The construction of these permutation schedules for both optimization problems can be done in polynomial time. It remains to check for both cases whether the buffer constraint is violated at any time. If an intermediate buffer is used, no buffer space is occupied at any time. In the case of a spanning buffer, at most one buffer unit is used for all cases of $\mathsf{bufUsage} \in \{s_J = a_J,\, s_J = 1\}$ which is a trivial lower bound for the buffer capacity Ω (or else it would not be possible to process any job in the case $\mathsf{bufType} = \mathit{spanningBuffer}$). □

The algorithm described in the proof of Theorem 5.1.9 can be directly used to efficiently solve the decision problem (for the considered special case) whether for two given numbers $B \geq 0$, $Q \geq 0$ there exists a feasible schedule σ that satisfies $C_{\max}(\sigma) \leq B$ and $R(\sigma) \geq Q$. In particular, it suffices to calculate $\ell_1 = \lfloor B/c \rfloor$, construct a permutation schedule σ^P with makespan $R(\sigma^P) = \ell_1 \cdot c$ that contains the ℓ_1 jobs with the highest profit and check whether $R(\sigma^P) \geq Q$ is satisfied.

The following lemma which describes the efficient solvability of a special case that is even more restricted, is later used in the tabular overview in Section 5.1.5.

Lemma 5.1.6. *Consider an instance of the Two-Stage VRP with Profits and Buffers where $c = 0$, $a_J = 0$ holds for all jobs J and $d_{ij} = 0$ holds for all edges e_{ij} in the graph. Then, a feasible permutation schedule can be constructed in polynomial time that is optimal for both optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$.*

Proof. Due to the restrictions, any permutation schedule σ^P that processes all jobs as early as possible has the makespan $C_{\max}(\sigma^P) = 0$, so it is possible to process all available jobs to obtain a solution σ^* that maximizes the total profit while having zero makespan. This schedule can be constructed in polynomial time. □

Existence of Permutation Schedules in the Set of Optimal Solutions

Recall Lemma 3.2.2 which states for Two-Machine Flow Shops with Buffers that any non-permutation schedule can be reordered to a permutation schedule without increase in makespan if the constant c is smaller than all processing times on M_1. The following results generalize this statement to the Two-Stage VRP with Profits and Buffers and show that this property holds even if non-zero travel times d_{ij} for the edges e_{ij} are allowed.

Lemma 5.1.7. *Let σ be a non-permutation schedule for an instance of the Two-Stage VRP with Profits and Buffers with the additional restriction that $a_J \geq c$ holds for all jobs J in the instance. Then, there exists a permutation schedule σ^P satisfying $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$, $R(\sigma^P) = R(\sigma)$ and $C_{\max}(\sigma^P) \leq C_{\max}(\sigma)$.*

Proof. Assume that $a_J \geq c$ holds for all jobs J. For a given non-permutation schedule σ, a permutation schedule σ^P is constructed as follows. Set $\pi(\sigma^P) = \pi^1(\sigma)$ so that both machines in σ^P process the jobs in the same order as M_1 in σ. Without loss of generality, it is assumed that ℓ out of the n jobs in the instance are processed in σ (i.e., $ProcessedJobs(\sigma) = \{J_1, J_2, \ldots, J_\ell\}$) and $\pi(\sigma^P) = \pi^1(\sigma) = (J_1, J_2, \ldots, J_\ell)$. Next, schedule the jobs on M_1 using the rule $S^1_{J_i}(\sigma^P) = S^1_{J_i}(\sigma)$ for all J_i (i.e., the M_1 starting times are the same as in σ). For M_2, all jobs J_i are schedules using the rule $S^2_{J_i}(\sigma^P) = C^1_{J_i}(\sigma^P)$.

It can be seen that $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$ and $R(\sigma^P) = R(\sigma)$ hold. The resulting schedule σ^P is a permutation schedule that satisfies $C_{\max}(\sigma^P) \leq$

$C_{\max}(\sigma)$, since both σ and σ^P need to process the job J_ℓ (the last job in $\pi^1(\sigma) = \pi(\sigma^P)$) on M_2, where J_ℓ is also the last job on M_2 in σ^P.

Next, it needs to be shown that σ^P is a valid schedule, i.e., that it is possible for M_2 to start all jobs at the specified times. In particular, M_2 must have finished the previous job and it must have reached the node containing the job to be processed. Formally, $S^2_{J_{i+1}}(\sigma^P) \geq C^2_{J_i}(\sigma^P) + d_{i,i+1}$ needs to hold (for $1 \leq i \leq \ell - 1$), as well as $S^2_{J_1}(\sigma^P) \geq d_{0,1}$. (Note that it is not necessary to check whether M_2 processes jobs that are not finished yet on M_1 due to the definition of σ^P).

This can be shown by induction. For the starting time $S^2_{J_1}(\sigma)$ in σ, the machine M_1 must have already traversed the edge $e_{0,1}$ (from the depot to the node that contains J_1) and processed J_1 at that time, i.e.. $S^2_{J_1}(\sigma) \geq a_{J_1} + d_{0,1} \geq d_{0,1}$. Thus, it is possible for M_2 to process J_1 at time $S^2_{J_1}(\sigma^P) = C^1_{J_1}(\sigma^P)$.

For the induction step, assume that J_i can be processed by M_2 in σ^P at the specified time $S^2_{J_i}(\sigma^P) = C^1_{J_i}(\sigma)$.Then, it is also possible for M_2 to process the subsequent job J_{i+1}, since $S^2_{J_{i+1}}(\sigma^P) = C^1_{J_{i+1}}(\sigma^P) \geq C^1_{J_1}(\sigma^P) + a_{J_{i+1}} + d_{i,i+1} = S^2_{J_i}(\sigma^P) + a_{J_{i+1}} + d_{i,i+1} \geq S^2_{J_i}(\sigma^P) + c + d_{i,i+1} = C^2_{J_i}(\sigma^P) + d_{i,i+1}$, or in short, $S^2_{J_{i+1}}(\sigma^P) \geq C^2_{J_i}(\sigma^P) + d_{i,i+1}$. By induction, the permutation Schedule σ^P is valid regarding the starting times.

It remains to check whether σ^P is feasible with respect to the buffer constraints. In the intermediate buffer model, the buffer is not used. Regarding the case with spanning buffer, note that at most 2 jobs J_i, J_{i+1} can enter the buffer in σ^P at any time due to the definition of σ^P. Assume that these two jobs exceed the buffer capacity, i.e., $s_{J_i} + s_{J_{i+1}} > \Omega$. Then it can be shown that in the original schedule σ the time interval $[C^1_{J_i}(\sigma),\ S^1_{J_{i+1}}(\sigma)]$ spans at least c time units, which is sufficiently large so that J_i and J_{i+1} cannot both be in the buffer in the constructed schedule σ^P.

To see this, assume that the interval $[C^1_{J_i}(\sigma),\ S^1_{J_{i+1}}(\sigma)]$ in σ spans less than c time units. In σ, the job J_i must have been processed on M_2 at some time before $S^1_{J_{i+1}}(\sigma)$, or else both jobs would enter the buffer. However, all possibilities for $S^2_{J_i}(\sigma)$ imply for the spanning buffer model that J_i and J_{i+1} are both in the buffer, which contradicts the assumption that they exceed the buffer capacity. Thus, when two jobs J_i, J_{i+1} cannot be in the buffer at the same time, they are not in the buffer at the same time in the constructed permutation schedule σ^P. □

However, an interesting asymmetry arises when the opposite case is considered where all jobs satisfy $a_J \leq c$. For the Two-Machine Flow Shop with Buffers, it

was shown that any non-permutation schedule can be reordered to a permutation schedule without increasing its makespan (see Lemma 3.2.1) for both cases when c is smaller or larger than all processing times a_J on M_1, but for the Two-Stage VRP with Profits and Buffers this only holds for the case when $a_J \geq c$ holds for all jobs J. For the case $a_J \leq c$, the example with 3 jobs given in the proof of Theorem 5.1.3 shows that even in this case that there exist non-permutation schedules with makespans that cannot be reached by any permutation schedule. Due to this, the same "gaps" in solution quality between non-permutation schedules and permutation schedules that were described in the previous section can potentially arise for this special case.

However, if the inequality $a_J \leq c$ is further strengthened by considering the extreme case where $a_J = 0$ holds for all jobs, the existence of optimal permutation schedules can be guaranteed due to the following lemma.

Lemma 5.1.8. *Let σ be a non-permutation schedule for an instance of the Two-Stage VRP with Profits and Buffers with the additional restriction that $a_J = 0$ holds for all jobs J in the instance. Then, there exists a permutation schedule σ^P satisfying $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$, $R(\sigma^P) = R(\sigma)$ and $C_{\max}(\sigma^P) \leq C_{\max}(\sigma)$.*

Proof. Similar to the proof of Lemma 5.1.7, but for a given non-permutation schedule σ that processes ℓ jobs the permutation $\pi^2(\sigma)$ is used for the permutation schedule σ^P, i.e., $\pi(\sigma^P) = \pi^2(\sigma)$. Without loss of generality, it is assumed that $\pi(\sigma^P) = (J_1, J_2, \ldots, J_\ell)$. All starting times in σ^P on M_2 are set to be the same as in σ (i.e., $S_J^2(\sigma^P) = S_J^2(\sigma)$ for all $J \in ProcessedJobs(\sigma)$) and for M_1 all jobs are scheduled using the rule $S_J^1(\sigma^P) = S_J^2(\sigma^P)$. By definition, σ^P satisfies $ProcessedJobs(\sigma^P) = ProcessedJobs(\sigma)$, $R(\sigma^P) = R(\sigma)$ and $C_{\max}(\sigma^P) = C_{\max}(\sigma)$, but it needs to be shown that it is possible for M_1 to process all of the jobs at the specified times and that the buffer constraint is not violated at any time.

The former property formally means that $S^1_{J_{i+1}} \geq C^1_{J_i} + d_{i,i+1}$ needs to hold for $1 \leq i \leq \ell - 1$ and $S^1_{J_1} \geq d_{0,1}$ needs to hold for the first job J_1. Note that it is not necessary to check whether M_2 processes jobs that are not finished yet on M_1 due to the definition of σ^P and since $a_J = 0$ holds for all jobs J. The aforementioned inequality can be shown by induction. At the starting time $S^2_{J_1}(\sigma)$ of the first job on M_2 in σ, machine M_2 must have already traveled to the node containing J_1, i.e.. $S^2_{J_1}(\sigma) \geq d_{0,1}$. Since the start time for J_1 in σ^P was chosen as $S^1_{J_1}(\sigma^P) = S^2_{J_1}(\sigma^P) = S^2_{J_1}(\sigma)$, it follows that $S^1_{J_1}(\sigma^P) \geq d_{0,1}$ holds which means that it is possible for M_1 to process J_1 at the

time $S^1_{J_1}(\sigma^P)$. Since $a_J = 0$, it holds that $C^1_{J_1}(\sigma^P) = S^1_{J_1}(\sigma^P) + a_J = S^1_{J_1}(\sigma^P) = S^2_{J_1}(\sigma^P)$.

For the induction step, assume that J_i can be processed by M_1 in σ^P at the time $S^1_{J_i}(\sigma^P) = S^2_{J_i}(\sigma)$, i.e., it holds that $S^2_{J_i}(\sigma^P) = C^1_{J_i}(\sigma^P)$. Using this equation shows that $S^1_{J_{i+1}}(\sigma^P) = S^2_{J_{i+1}}(\sigma^P) \geq S^2_{J_i}(\sigma^P) + c + d_{i,i+1} = C^1_{J_i}(\sigma^P) + d_{i,i+1} + c \geq C^1_{J_i}(\sigma^P) + d_{i,i+1}$, so it is possible for M_1 to process J_{i+1}. By induction, the schedule σ^P is valid regarding its starting times.

In addition, the buffer constraint is not violated at any time: If $s_J = a_J$ holds for all jobs J, no job takes up any buffer space. If $s_J = 1$, the buffer is not used in the case $\mathsf{bufType} = \textit{intermediateBuffer}$, whereas for $\mathsf{bufType} = \textit{spanningBuffer}$ at most one job can be in the buffer at any time, so that at most one buffer unit is used. The buffer capacity Ω can be assumed to be at least $\Omega \geq 1$, or else it is not possible for any job to be processed without violating the buffer constraint. □

The Gap Between Permutation Schedules an Non-Permutation Schedules

For the Orienteering Problem, which is a restricted case of the Two-Stage VRP with Profits and Buffers where all processing times are zero, it was shown in Section 4.2 that any non-permutation schedule can be converted into a permutation schedule without increasing its length (see Lemma 4.2.1). From this it directly follows that the set of optimal schedules always contains a permutation schedule for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ (see Corollary 4.2.1), but also for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ which originally is not part of the Orienteering Problem.

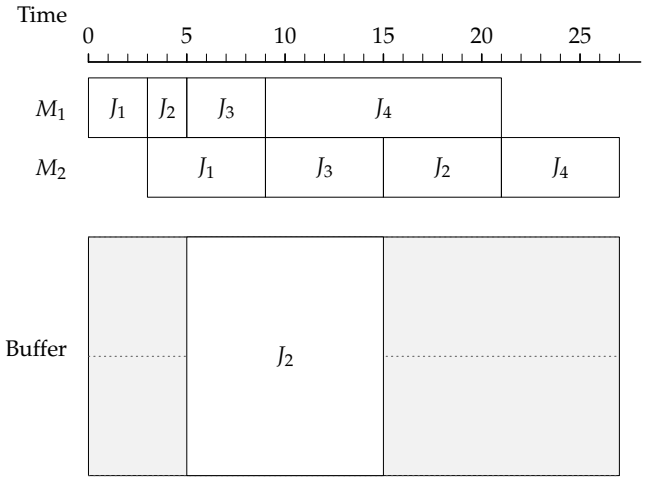
For the Two-Machine Flow Shop with Buffers (the restricted case where all edges have travel time $d_{ij} = 0$) on the other hand, a "gap" in solution quality between permutation schedules and non-permutation schedules was found when $s_J = a_J$ holds for all jobs J (see Section 3.2.3). However, since the Two-Machine Flow Shop with Buffers deals with the problem of minimizing makespan (and thus, with the optimization problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$), the other optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ was not considered in that section. But in the context of the Two-Stage VRP with Profits and Buffers, the question of how large the solution quality gap is for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ naturally arises for the restricted case with zero travel times on all edges.

In the following, let $R(\sigma^*, \mathcal{I})$ be the total collected profit of a schedule σ^* that is optimal on instance $\mathcal{I}$ with respect to $\mathcal{O}_{\max R,\, C_{\max} \leq B}$. Let $R(\sigma^*_{perm}, \mathcal{I})$ be the total profit of a "best" permutation schedule σ^*_{perm}, i.e., a permutation schedule that

Figure 5.1: Example instance for Lemma 5.1.9. The left side shows the jobs of that instance, their processing times (a_{J_i} and b_{J_i}) and how much buffer space s_{J_i} they occupy. In addition, the buffer capacity Ω is shown at the bottom left. The right side shows an example for a non-permutation schedule σ^* that minimizes the makespan $C_{\max}$.

Job	a_{J_i}	b_{J_i}	s_{J_i}
J_1	3	6	3
J_2	2	6	2
J_3	4	6	4
J_4	12	6	12

$\Omega = 2$



maximizes the total profit out of all permutation schedules on an instance $\mathcal{I}$. It is assumed that $s_J = a_J$ holds for all jobs in the following. In order to investigate the "gap" for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, two straightforward statements are given first that demonstrate concrete values for the solution quality ratio $R(\sigma^*, \mathcal{I}) / R(\sigma^*_{perm}, \mathcal{I})$.

Lemma 5.1.9. *There exists an instance $\mathcal{I}$ of the Two-Stage VRP with Profits and Buffers where $d_{ij} = 0$ holds for all edges e_{ij}, $s_J = a_J$ holds for all jobs J and* bufType $=$ *intermediateBuffer such that*

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} = \frac{4}{3}.$$

Proof. Recall Theorem 3.2.5 which states that for every $n > 3$ with $n \in \mathbb{N}$ there exists an instance where the set of optimal schedules (with respect to the makespan minimization problem) that contains no permutation schedule. Take one such instance $\mathcal{I}$ for $n = 4$, for example the one shown in Figure 5.1 taken from [57] (for a proof on why the set of makespan-minimizing schedules contains no permutation schedule, see [57]). The travel times d_{ij} are assumed to be zero for all edges.

For this instance, there exists an optimal (non-permutation) schedule σ^* with makespan $C_{\max}(\sigma^*)$ that cannot be reached by any permutation schedule. Re-

garding the example in Figure 5.1, $C_{\max}(\sigma^*) = 27$. For the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, the budget B for the budget constraint $C_{\max}(\sigma) \leq B$ can be chosen as $B = C_{\max}(\sigma^*)$ so that it is not possible for a permutation schedule σ^P to process all 4 jobs of the instance without violating the budget constraints. Thus, a permutation schedule that does not violate that constraint can only process 3 jobs at most. Note that the existence of permutation schedules that process 3 jobs without violating the budget constraint is guaranteed, since there trivially exists a (non-permutation) schedule that processes 3 jobs and any non-permutation schedule with 3 jobs can be converted to a permutation schedule without increasing its length due to Theorem 3.2.5. By choosing the job profits as $r_J = 1$ for all jobs, it immediately follows for this instance that the solution quality ratio is $R(\sigma^*, \mathcal{I})/R(\sigma^*_{perm}, \mathcal{I}) = 4/3$. □

The same idea can be applied to the case with the same restrictions and $\mathsf{bufType} = \mathit{spanningBuffer}$. In the case with spanning buffer, the existence of permutation schedules in the set of makespan-minimizing schedules is guaranteed for instances with up to $n = 6$ jobs, whereas for every $n > 6$ there exists an instance where the set of schedules that minimize makespan contains no permutation schedule (see Theorem 3.2.4). The same idea used in the proof of Lemma 5.1.9 can be applied to this case to straightforwardly show the following statement.

Lemma 5.1.10. *There exists an instance $\mathcal{I}$ of the Two-Stage VRP with Profits and Buffers where $d_{ij} = 0$ holds for all edges e_{ij}, $s_J = a_J$ holds for all jobs J and* $\mathsf{bufType} = \mathit{spanningBuffer}$ *such that*

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} = \frac{7}{6}.$$

After presenting concrete values that can be reached for the solution quality ratio $R(\sigma^*, \mathcal{I})/R(\sigma^*_{perm}, \mathcal{I})$, it is investigated in the following how large this ratio (and thus, the "gap" between non-permutation schedules and permutation schedules) can potentially be. For this, a lemma is stated first that provides additional information for analyzing the gap. This property can be shown due to the restriction $d_{ij} = 0$ for all edges e_{ij}.

Lemma 5.1.11. *Let σ be a schedule for an instance of the Two-Stage VRP with Profits and Buffers where all edges have travel time $d_{ij} = 0$ and all jobs J satisfy $s_J = a_J$. Let $\ell = |\mathit{ProcessedJobs}(\sigma)|$ and consider the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ with budget*

B. Then, there exists a permutation schedule σ^P satisfying $C_{\max}(\sigma^P) \leq B$ where the total profit $R(\sigma^P)$ is not less than the sum of the profits of the $\lfloor \ell/2 \rfloor$ jobs in ProcessedJobs(σ) with the lowest profit.

Proof. Without loss of generality, let *ProcessedJobs*$(\sigma) = \{J_1, J_2, \ldots, J_\ell\}$, and denote the profit values corresponding to these jobs as $r_{J_1}, r_{J_2}, \ldots, r_{J_\ell}$ with $r_{J_1} \leq r_{J_1} \leq \cdots \leq r_{J_\ell}$. A permutation schedule σ^P with the desired properties is constructed as follows. Define $\ell_1 = \lfloor \ell/2 \rfloor$ and take the ℓ_1 jobs with the shortest processing time on M_1. These jobs are processed in σ^P such that the processing times between different jobs do not overlap (similar to the trivial permutation schedule constructed in the proof of Theorem 3.2.8). Formally, $S^1_{J_1}(\sigma^P) = 0$ and $S^2_{J_1}(\sigma^P) = a_{J_1}$ for the first job J_1 as well as $S^1_{J_i}(\sigma^P) = C^2_{J_{i-1}}(\sigma^P)$ and $S^2_{J_i}(\sigma^P) = S^1_{J_i}(\sigma^P) + a_{J_i} = C^1_{J_i}(\sigma^P)$ for $i = 2, 3, \ldots, \ell_1$.

Since $d_{ij} = 0$ for all edges e_{ij}, it is not hard to see that σ^P is a valid schedule (i.e., it is possible for M_1 and M_2 to process all jobs at the specified times). Furthermore, the buffer constraint is not violated at any time: In the case bufType $=$ *intermediateBuffer*, the buffer is not used at any time. If bufType $=$ *spanningBuffer*, not more than $\max_{J \in \mathit{ProcessedJobs}(\sigma)} s_J$ units of buffer are used, which is a trivial lower bound for the buffer capacity Ω (otherwise the original schedule σ would also violate the buffer constraint). Thus, σ^P is also feasible with respect to the buffer constraint. Regarding the total profit $R(\sigma^P)$, note that $\ell_1 = \lfloor \ell/2 \rfloor$ jobs are processed in σ^P so that the sum $\sum_{J \in \mathit{ProcessedJobs}(\sigma^P)} r_J$ of their profits cannot be smaller than $r_{J_1} + r_{J_2} + \cdots + r_{\ell_1}$ (the sum of the ℓ_1 lowest profit values).

It remains to show that σ^P does not violate the buffer constraint $C_{\max}(\sigma^P) \leq B$. To see this, let $A(\sigma) = \sum_{J \in \mathit{ProcessedJobs}(\sigma)} a_J$ and $A(\sigma^P) = \sum_{J \in \mathit{ProcessedJobs}(\sigma^P)} a_J$. Note that σ^P processes the $\ell_1 = \lfloor \ell/2 \rfloor$ jobs with the shortest processing time on M_1 so that $A(\sigma^P) \leq \frac{1}{2} A(\sigma)$ holds. Due to this, $C_{\max}(\sigma^P) = A(\sigma^P) + \ell_1 c \leq \frac{1}{2} A(\sigma) + \frac{1}{2} \ell c$. A trivial lower bound for the makespan of the original schedule σ is $C_{\max}(\sigma) \geq \max\{A(\sigma),\ \ell c\}$. In addition, $C_{\max}(\sigma) \leq B$ needs to hold, or else σ would violate the budget constraint. From these inequalities, it follows that

$$\begin{aligned} C_{\max}(\sigma^P) &\leq \frac{1}{2} A(\sigma) + \frac{1}{2} \ell c \leq \frac{1}{2} \max\{A(\sigma),\ \ell c\} + \frac{1}{2} \max\{A(\sigma),\ \ell c\} \\ &= \max\{A(\sigma),\ \ell c\} \leq C_{\max}(\sigma) \leq B. \end{aligned}$$

□

In order to consider both cases $\mathsf{bufType} = \textit{intermediateBuffer}$ and $\mathsf{bufType} = \textit{spanningBuffer}$ at the same time, a variable k_0 is introduced in the following, which, for an instance of the Two-Stage VRP with Profits and Buffers where all edges have travel time $d_{ij} = 0$ (similar to a Two-Machine Flow Shop with Buffers), describes the number of jobs in an instance until which the existence of permutation schedules in the set of makespan-minimizing schedules is guaranteed. In other words, k_0 is the highest number where for any non-permutation schedule σ with $|\textit{ProcessedJobs}(\sigma)| \leq k_0$ there exists a permutation schedule σ^P that processes the same jobs without taking more time, i.e., $C_{\max}(\sigma^P) \leq C_{\max}(\sigma)$. For the considered case of the Two-Stage VRP with Profits and Buffers and the considered buffer types $\mathsf{bufType} \in \{\textit{intermediateBuffer}, \textit{spanningBuffer}\}$, it can be said that $k_0 = 3$ if $\mathsf{bufType} = \textit{intermediateBuffer}$ (see Theorem 3.2.5) and $k_0 = 6$ if $\mathsf{bufType} = \textit{spanningBuffer}$ (see Theorem 3.2.4) .

The following lemma provides an additional property that is later used when analyzing the solution quality gap between non-permutation schedules and permutation schedules.

Lemma 5.1.12. *Let $\mathcal{I}$ be an instance of the Two-Stage VRP with Profits and Buffers where (i) $d_{ij} = 0$ holds for all edges e_{ij}, (ii) $s_J = a_J$ holds for all jobs and (iii) all non-permutation schedules processing not more than k_0 jobs can be transformed into feasible permutation schedules without increasing their length. Let σ be a schedule for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ with budget B that processes ℓ jobs. Then, there exists a permutation schedule σ^P satisfying $C_{\max}(\sigma^P) \leq B$ where its total profit $R(\sigma^P)$ is not smaller than the sum of the k_0 highest profit values of the jobs in $\textit{ProcessedJobs}(\sigma)$.*

Proof. Without loss of generality, it is assumed that $\textit{ProcessedJobs}(\sigma) = \{J_1, J_2, \ldots, J_\ell\}$ and that $r_{J_1}, r_{J_2}, \ldots, r_{J_\ell}$ are the profit values of the jobs $J_1, J_2, \ldots, J_\ell$ with $r_{J_1} \leq r_{J_2} \leq \cdots \leq r_{J_\ell}$. Define $\mathcal{J}_{k_0} = \{J_\ell, J_{\ell-1}, J_{\ell-2}, \ldots, J_{\ell-(k_0-1)}\}$ as the set that contains the k_0 jobs with the highest profit. A permutation schedule σ^P with the desired properties can be constructed from σ as follows. First, construct a temporary schedule $\tilde{\sigma}$ by removing all jobs from σ except the jobs in $\mathcal{J}_{k_0}$ and use the same starting times as in σ for the remaining jobs: $S_J^1(\tilde{\sigma}) = S_J^1(\sigma)$ and $S_J^2(\tilde{\sigma}) = S_J^2(\sigma)$ for all $J \in \mathcal{J}_{k_0}$.

It is not possible for $\tilde{\sigma}$ to violate the buffer constraint or budget constraint. Since $\tilde{\sigma}$ processes k_0 jobs, it is possible to reorder $\tilde{\sigma}$ into a permutation schedule σ^P without increasing the length of the schedule. Thus, σ^P cannot violate any constraints either

and it processes the k_0 jobs with the highest profit contained in $\mathit{ProcessedJobs}(\sigma)$. □

From Lemma 5.1.11 and Lemma 5.1.12, it follows that the "best" permutation schedule σ^*_{perm} for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ (i.e., the permutation schedule that maximizes the profit from the set of all permutation schedules that do not violate the budget constraint) cannot obtain a profit that is less than the values established in the lemmata above. This information is now used to establish an upper bound for the solution quality ratio $R(\sigma^*)/R(\sigma^*_{perm})$.

Lemma 5.1.13. *Let $\mathcal{I}$ be an instance of the Two-Stage VRP with Profits and Buffers where (i) $d_{ij} = 0$ holds for all edges e_{ij}, (ii) $s_J = a_J$ holds for all jobs J and (iii) all non-permutation schedules processing not more than k_0 jobs can be transformed into feasible permutation schedules without increasing their length. Let σ^* be an optimal schedule for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ with budget B. Let $\ell = |\mathit{ProcessedJobs}(\sigma^*)|$ and let σ^*_{perm} be a permutation schedule that maximizes the score out of all permutation schedules σ^P that satisfy $C_{\max}(\sigma^P) \leq B$. Then, the solution quality ratio $R(\sigma^*, \mathcal{I})/R(\sigma^*_{perm}, \mathcal{I})$ is bounded from above by*

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} \leq 1 + \frac{\lceil n/2 \rceil}{k_0},$$

if $k_0 < \lceil \ell/2 \rceil$ holds. If $k_0 \geq \lceil \ell/2 \rceil$, the solution quality ratio is bounded from above by 2.

Proof. Without loss of generality, it is assumed for an optimal schedule σ^* that $\mathit{ProcessedJobs}(\sigma^*) = \{J_1, J_2, \ldots, J_\ell\}$. Let $R = R(\sigma^*, \mathcal{I}) > 0$ be the total profit collected in σ^* on an instance $\mathcal{I}$ with the properties mentioned above and let $r_1, r_2, \ldots, r_\ell$ be the profit values of the jobs $J_1, J_2, \ldots, J_\ell$ with $r_1 \leq r_2 \leq \cdots \leq r_\ell$. Due to Lemma 5.1.11 and Lemma 5.1.12, a "best" permutation schedule σ^*_{perm} must obtain a value that is not less than $R_* = r_1 + r_2 + \cdots + r_{\lfloor \ell/2 \rfloor}$ or $R^* = r_\ell + r_{\ell-1} + r_{\ell-2} + \cdots + r_{\ell-(k_0-1)}$. In short, $R(\sigma^*_{perm}, \mathcal{I}) \geq \max\{R^*, R_*\}$, so that the solution quality ratio is bounded by

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} \leq \frac{R}{\max\{R^*, R_*\}}$$

Since the total profit of an optimal schedule σ^* does not change, it can be assumed that R is constant, so the upper bound for that ratio is maximized when $\max\{R^*, R_*\}$ is minimized.

In the following, it is investigated how the total profit R should be distributed on $r_1, r_2, \ldots, r_\ell$ so that $\max\{R^*, R_*\}$ is minimized. This problem can be modeled

as a linear program with decision variables $r_1, r_2, \ldots, r_\ell$. Note that the following linear program is expressed as a maximization problem with the goal of maximizing $-\max\{R^*, R_*\}$ and that this term can be expressed as a linear function by introducing a temporary variable Z and adding the constraints $R_* \leq Z$ and $R^* \leq Z$:

$$\text{Maximize} \quad f(r_1, r_2, \ldots, r_n, Z) = -Z$$

with $r_1, r_2, \ldots, r_n, Z \geq 0$ subject to the constraints

$$\begin{aligned}
(y_i) \quad & r_i - r_{i+1} && \leq 0 \quad (i = 1, 2, \ldots, \ell - 1) \\
(y_o) \quad & r_{\ell-(k_0-1)} + \cdots + r_{\ell-1} + r_\ell && \leq Z \\
(y_u) \quad & r_1 + r_2 + \cdots + r_{\lfloor \ell/2 \rfloor} && \leq Z \\
(y_s) \quad & r_1 + r_2 + \cdots + r_\ell && = R,
\end{aligned}$$

where the terms "(y)" are used as names for the constraints. This optimization problem contains $\ell + 1$ variables, $(\ell - 1) + 2$ inequality constraints and one equality constraint.

Recall the proof of Theorem 5.1.6 where the duality theorems for linear programming were used in the following to construct a solution $\underline{r} = (r_1, r_2, \ldots, r_{\ell-1}, r_\ell)$ and establish its optimality. In short, these theorems state that it suffices to present a feasible solution $\underline{y}$ for the dual linear program with objective function g that satisfies $g(\underline{y}) = f(\underline{r})$ in order to show that $\underline{r}$ is an optimal solution that maximizes f. This approach is also used in the following, but the verification that $\underline{y}$ is a feasible solution will require further analysis which is done later in the proof. First, the dual linear program for the linear program given above (the "primal linear program") is presented. Note that constraints y and variables r of the primal linear program become variables and constraints in the dual program, respectively (see, e.g., [25, 53] for details on how to construct dual linear programs):

$$\text{Minimize} \quad g(y_1, y_2, \ldots, y_{\ell-1}, y_o, y_u, y_s) = R \cdot y_s$$

with $y_1, y_2, \ldots, y_\ell, y_o, y_u \geq 0$ and $y_s \in \mathbb{R}$ subject to the constraints

$$\begin{aligned}
(r_1) \quad & y_1 + y_u + y_s && \geq 0 \\
(r_2) \quad & -y_1 + y_2 + y_u + y_s && \geq 0 \\
(r_3) \quad & -y_2 + y_3 + y_u + y_s && \geq 0 \\
& \ldots \\
(r_{\lfloor \ell/2 \rfloor}) \quad & -y_{\lfloor \ell/2 \rfloor - 1} + y_{\lfloor \ell/2 \rfloor} + y_u + y_s && \geq 0 \\
(r_{\lfloor \ell/2 \rfloor + 1}) \quad & -y_{\lfloor \ell/2 \rfloor} + y_{\lfloor \ell/2 \rfloor + 1} + y_s && \geq 0 \\
(r_{\lfloor \ell/2 \rfloor + 2}) \quad & -y_{\lfloor \ell/2 \rfloor + 1} + y_{\lfloor \ell/2 \rfloor + 2} + y_s && \geq 0 \\
& \ldots \\
(r_{\ell - k_0}) \quad & -y_{\ell - k_0 - 1} + y_{\ell - k_0} + y_s && \geq 0 \\
(r_{\ell - (k_0 - 1)}) \quad & -y_{\ell - k_0} + y_{\ell - k_0 + 1} + y_o + y_s && \geq 0 \\
(r_{\ell - (k_0 - 2)}) \quad & -y_{\ell - k_0 + 1} + y_{\ell - k_0 + 2} + y_o + y_s && \geq 0 \\
& \ldots \\
(r_{\ell - 1}) \quad & -y_{\ell - 2} + y_{\ell - 1} + y_o + y_s && \geq 0 \\
(r_\ell) \quad & -y_{\ell - 1} + y_o + y_s && \geq 0 \\
(Z) \quad & -y_o - y_u && \geq -1,
\end{aligned}$$

where the terms in parentheses on the left are the names for the constraints of the dual linear program. The dual linear program has $(\ell - 1) + 3$ variables and $\ell + 1$ constraints.

In the following, the two cases $k_0 < \lceil \ell/2 \rceil$ and $k_0 \geq \lceil \ell/2 \rceil$ named in the statement of the theorem are analyzed separately.

First case: $k_0 < \lceil \ell/2 \rceil$**.** We partition the index set $I = \{1, 2, \ldots, \ell\}$ into three sets $I_1 = \{1, 2, \ldots, \lfloor \ell/2 \rfloor\}$, $I_2 = \{\lfloor \ell/2 \rfloor + 1, \lfloor \ell/2 \rfloor + 2, \ldots, \ell - k_0\}$, $I_3 = \{\ell - (k_0 - 1), \ell - (k_0 - 2), \ldots, \ell - 1, \ell\}$. The choice of these sets is based on the idea that $\sum_{i \in I_1} r_i = R_*$ and $\sum_{i \in I_3} r_i = R^*$, whereas I_2 contains the remaining indices.

Consider the following solution $\underline{r}^* = (r_1, r_2, \ldots, r_\ell)$: For $i < \lfloor \ell/2 \rfloor - (k_0 - 1)$, all r_i are set to $r_i = 0$ and the total profit R is equally distributed over the remaining profit values. By doing so, only the k_0 most valuable profits r_i for $i \in I_1$ have a profit $r_i \neq 0$. Since I_3 contains k_0 elements, it follows that the number of non-zero profits

r_i in $i \in I_3$ is also k_0 and that

$$\sum_{i \in I_1} r_i = \sum_{i \in I_3} r_i.$$

Due to the assumption $k_0 < \lceil \ell/2 \rceil$, it is guaranteed that there exists at least one profit value r_i, $i \in I_1$ with $r_i = 0$.

Let q be the total number of profit values r_i $(i \in I)$ in the solution $\underline{r}^*$ that have a value $r_i \neq 0$. In order to determine q, two cases are considered. If ℓ is even, then

$$\begin{aligned} q &= k_0 + \underbrace{(\ell - k_0) - (\lfloor \ell/2 \rfloor + 1) + 1}_{=|I_2|} + k_0 \\ &= \ell/2 + k_0 \end{aligned}$$

and it can be seen that $|I_2| = \ell/2 - k_0$.

If ℓ is an odd number, then $q = (\ell + 1)/2 + k_0$ (to see this, recall that for $i < \lfloor \ell/2 \rfloor - (k_0 - 1)$ all r_i are set to $r_i = 0$, so all r_i for larger i have a value $r_i \neq 0$) and $|I_2| = \ell/2 - k_0 + 1/2$. Both cases can be summarized as

$$q = \lceil \ell/2 \rceil + k_0$$

and $|I_2| = \lceil \ell/2 \rceil - k_0$. It follows that the r_i with $r_i \neq 0$ have the value $r_i = R/q$.

For the constructed solution $\underline{r}^*$ (of the primal linear program), it follows that $R^* = R_* = k_0 \cdot R/q$, so that the temporary variable Z introduced above can be set as $Z = k_0 R/q$ from which $f(\underline{r}^*, Z) = -k_0 R/q$ follows. It is straightforward to check that $\underline{r}^*$ and Z satisfy all constraints.

Next, a feasible solution $\underline{y}^*$ for the dual linear program is constructed such that $g(\underline{y}^*) = R \cdot y_s = f(\underline{r}^*, Z) = -k_0 R/q$. Since the objective function g only depends on y_s, it is not hard to see that y_s must have the value

$$y_s = -\frac{k_0}{q} = -\frac{k_0}{\lceil \ell/2 \rceil + k_0}.$$

Regarding y_u, the value

$$y_u = -y_s = \frac{k_0}{\lceil \ell/2 \rceil + k_0}$$

is chosen and $y_1 = y_2 = \cdots = y_{\lfloor \ell/2 \rfloor} = 0$. With these values, the constraints

$(r_1), (r_2), \ldots, (r_{\lfloor \ell/2 \rfloor})$ are satisfied. For y_o, we set

$$y_o = 1 - y_u = \frac{\lceil \ell/2 \rceil + k_0 - k_0}{\lceil \ell/2 \rceil + k_0} = \frac{\lceil \ell/2 \rceil}{\lceil \ell/2 \rceil + k_0},$$

so that constraint (Z) is satisfied.

Regarding the constraints $(r_{\lfloor \ell/2 \rfloor+1})$, $(r_{\lfloor \ell/2 \rfloor+2}), \ldots, (r_{\ell-k_0})$, i.e., the constraints that contain neither y_u nor y_o, the values

$$y_{\lfloor \ell/2 \rfloor+i} = -y_s \cdot i = \frac{i}{\lceil \ell/2 \rceil + k_0} \cdot k_0$$

are chosen for $i = 1, 2, \ldots, |I_2|$, where $|I_2|$ was shown above to be $|I_2| = \lceil \ell/2 \rceil - k_0$. In particular for the case $i = |I_2| = \lceil \ell/2 \rceil - k_0$, recall that the index $\lfloor \ell/2 \rfloor + i$ of $y_{\lfloor \ell/2 \rfloor+i}$ is equivalent to $\lfloor \ell/2 \rfloor + |I_2| = \ell - k_0$, so that

$$y_{\ell-k_0} = \frac{\lceil \ell/2 \rceil - k_0}{\lceil \ell/2 \rceil + k_0} \cdot k_0.$$

In the following, the abbreviation $K = \frac{\lceil \ell/2 \rceil - k_0}{\lceil \ell/2 \rceil + k_0}$ is used. Due to the assumption $k_0 < \lceil \ell/2 \rceil$ made above, $K \geq 0$ holds, so that $y_{\ell-k_0} \geq 0$. By inserting the values chosen so far into constraint $(r_{\ell-(k_0-1)})$ and simplifying the terms, one obtains

$$\begin{aligned} & -y_{\ell-k_0} + y_{\ell-k_0+1} + y_o + y_s \geq 0 \\ \Leftrightarrow \quad & -Ky_0 + y_{\ell-k_0+1} + \frac{\lceil \ell/2 \rceil}{\lceil \ell/2 \rceil + k_0} - \frac{k_0}{\lceil \ell/2 \rceil + k_0} \geq 0 \\ \Leftrightarrow \quad & K(1-k_0) + y_{\ell-k_0+1} \geq 0, \end{aligned}$$

which means that this constraint is satisfied if and only if $y_{\ell-k_0+1} \geq K(k_0 - 1)$.

We choose the value of $y_{\ell-k_0+1}$ as $y_{\ell-k_0+1} = K(k_0 - 1)$ and consider the next

constraint $(r_{\ell-(k_0-2)})$:

$$\begin{aligned} & -y_{\ell-k_0+1} + y_{\ell-k_0+2} + y_o + y_s \geq 0 \\ \Leftrightarrow \quad & K(1-k_0) + y_{\ell-k_0+2} + \frac{\lceil \ell/2 \rceil}{\lceil \ell/2 \rceil + k_0} - \frac{k_0}{\lceil \ell/2 \rceil + k_0} \geq 0 \\ \Leftrightarrow \quad & K(2-k_0) + y_{\ell-k_0+2} \geq 0 \end{aligned}$$

Similar to the previous constraint, $y_{\ell-k_0+2}$ is chosen as $y_{\ell-k_0+2} = -K(k_0-2)$ in order to satisfy constraint $(r_{\ell-(k_0-2)})$. This argument can be iteratively repeated for the constraints $(r_{\ell-(k_0-h)})$ with $h = 1, 2, \ldots, k_0 - 1$ where the values $y_{\ell-k_0+h}$ are chosen as

$$y_{\ell-k_0+h} = K(k_0 - h).$$

These resulting values are non-negative and satisfy constraints $(r_{\ell-(k_0-h)})$ with $h = 1, 2, \ldots, k_0 - 1$. Note that the case $h = k_0 - 1$ corresponds to constraint $(r_{\ell-1})$. Finally, consider constraint (r_ℓ) with the values chosen so far:

$$\begin{aligned} & -y_{\ell-1} + y_o + y_s \geq 0 \\ \Leftrightarrow \quad & K((k_0-1)-k_0) + \frac{\lceil \ell/2 \rceil}{\lceil \ell/2 \rceil + k_0} - \frac{k_0}{\lceil \ell/2 \rceil + k_0} \geq 0 \\ \Leftrightarrow \quad & K \cdot (-1) + K \geq 0 \\ \Leftrightarrow \quad & 0 \geq 0 \end{aligned}$$

This shows that (r_ℓ) is satisfied.

To summarize, the solution $\underline{y}^*$ with the values

$$\begin{aligned} y_s &= -\frac{k_0}{\lceil \ell/2 \rceil + k_0} \\ y_u &= \frac{k_0}{\lceil \ell/2 \rceil + k_0} \\ y_o &= \frac{\lceil \ell/2 \rceil}{\lceil \ell/2 \rceil + k_0} \\ y_1 &= 0 \\ y_2 &= 0 \\ &\dots \\ y_{\lfloor \ell/2 \rfloor} &= 0 \end{aligned}$$

$$\begin{aligned} y_{\lfloor \ell/2 \rfloor + 1} &= \frac{1}{\lceil \ell/2 \rceil + k_0} \cdot k_0 \\ y_{\lfloor \ell/2 \rfloor + 2} &= \frac{2}{\lceil \ell/2 \rceil + k_0} \cdot k_0 \\ &\dots \\ y_{\ell - k_0} &= \frac{\lceil \ell/2 \rceil - k_0}{\lceil \ell/2 \rceil + k_0} \cdot k_0 \qquad (\text{corresponds to } y_{\lfloor \ell/2 \rfloor + (\lceil \ell/2 \rceil - k_0)}) \\ y_{\ell - k_0 + 1} &= \frac{\lceil \ell/2 \rceil - k_0}{\lceil \ell/2 \rceil + k_0} \cdot (k_0 - 1) \\ y_{\ell - k_0 + 2} &= \frac{\lceil \ell/2 \rceil - k_0}{\lceil \ell/2 \rceil + k_0} \cdot (k_0 - 2) \\ &\dots \\ y_{\ell - 1} &= \frac{\lceil \ell/2 \rceil - k_0}{\lceil \ell/2 \rceil + k_0} \cdot 1 \qquad (\text{corresponds to } y_{\ell - k_0 + (k_0 - 1)}) \end{aligned}$$

satisfies all constraints and obtains the value $g(\underline{y}^*) = f(\underline{r}^*, Z) = -k_0 R/q$, thus proving that $\underline{r}^*$ is an optimal solution. Since in the primal linear program Z corresponds to $\max\{R_*, R^*\}$ and $Z = k_0 R/q$ with $q = \lceil \ell/2 \rceil + k_0$, the solution quality ratio in the case $k_0 < \lceil \ell/2 \rceil$ can be bounded from above by

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} \leq \frac{R}{\max\{R^*, R_*\}} = \frac{R}{\frac{k_0 R}{\lceil \ell/2 \rceil + k_0}} = \frac{\lceil \ell/2 \rceil + k_0}{k_0} = 1 + \frac{\lceil \ell/2 \rceil}{k_0}.$$

Second case $k_0 \geq \lceil \ell/2 \rceil$. For this case it is not possible to define the index set I_2 and q as in the first case, so a different solution $\underline{r}^*$ needs to be constructed. Consider the solution $\underline{r}^*$ where all profit values have the same value $r_i = R/\ell$ (for $i = 1, 2, \ldots, \ell$). It is straightforward to check that this solution satisfies all constraints of the primal linear program. In this case, $R_* = k_0 \cdot R/\ell \geq \lfloor \ell/2 \rfloor \cdot R/\ell = R^*$, so that Z can be chosen as $Z = k_0 \cdot R/\ell$ leading to the objective value $f(\underline{r}^*, Z) = -k_0 R/\ell$.

Next, a feasible solution $\underline{y}^*$ for the dual linear program is constructed that satisfies $g(\underline{y}^*) = f(\underline{r}^*, Z) = -k_0 R/\ell$. Since $g(\underline{y}^*) = Ry_s$ only depends on y_s, the value of y_s must be chosen as $y_s = -k_0/\ell$. By setting $y_o = 1$ and $y_u = 0$, constraint (Z) is satisfied. Next, values for $\underline{y}^*$ need to be set so that the constraints which do not contain y_o are satisfied, i.e., constraints $(r_1), (r_2), \ldots, (r_{\ell - k_0})$. This is straightforward to do by starting with $y_1 = -y_s$, followed by $y_i = -y_s \cdot i$ for $i = 1, 2, \ldots, \ell - k_0$.

Consider constraint $(r_{\ell-(k_0-1)})$:

$$\begin{aligned} & -y_{\ell-k_0} + y_{\ell-k_0+1} + y_o + y_s \geq 0 \\ \Leftrightarrow \quad & -\frac{k_0}{\ell}(\ell - k_0) + y_{\ell-k_0+1} + \frac{\ell}{\ell} - \frac{k_0}{\ell} \geq 0 \\ \Leftrightarrow \quad & \frac{\ell - k_0}{\ell}(1 - k_0) + y_{\ell-k_0+1} \geq 0 \end{aligned}$$

By setting $y_{\ell-k_0+1} = -\frac{\ell-k_0}{\ell}(1-k_0)$, this constraint is satisfied. By repeating this argument for the following constraints (similar to the first case), the next values in $\underline{y}^*$ are chosen as

$$y_{\ell-k_0+h} = -\frac{\ell - k_0}{\ell}(h - k_0)$$

for $h = 1, 2, \ldots, k_0 - 1$. With the values chosen so far, all constraints up to $(r_{\ell-1})$ are satisfied. It remains to check constraint (r_ℓ):

$$\begin{aligned} & -y_{\ell-1} + y_o + y_s \geq 0 \\ \Leftrightarrow \quad & \frac{\ell - k_0}{\ell}(-1) + \frac{\ell}{\ell} - \frac{k_0}{\ell} \geq 0 \\ \Leftrightarrow \quad & 0 \geq 0 \end{aligned}$$

To summarize, the solution $\underline{y}^*$ with the values

$$\begin{aligned}
y_s &= -\frac{k_0}{\ell} \\
y_u &= 0 \\
y_o &= 1 \\
y_1 &= \frac{1}{\ell} \cdot k_0 \\
y_2 &= \frac{2}{\ell} \cdot k_0 \\
&\dots \\
y_{\ell-k_0} &= \frac{\ell - k_0}{\ell} \cdot k_0 \\
y_{\ell-k_0+1} &= \frac{\ell - k_0}{\ell} \cdot (k_0 - 1) \\
y_{\ell-k_0+2} &= \frac{\ell - k_0}{\ell} \cdot (k_0 - 2) \\
&\dots \\
y_{\ell-1} &= \frac{\ell - k_0}{\ell} \cdot 1 \qquad \text{(corresponds to } y_{\ell-k_0+(k_0-1)}\text{)}
\end{aligned}$$

satisfies all constraints and obtains the value $g(\underline{y}^*) = f(\underline{r}^*, Z) = -k_0 R/\ell$, thus proving that $\underline{r}^*$ is an optimal solution. Using the values of $\underline{r}^*$ obtained for this case leads to the upper bound

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} \leq \frac{R}{\max\{R^*, R_*\}} = \frac{R}{k_0 \cdot \frac{R}{\ell}} = \frac{\ell}{k_0}$$

for the solution quality ratio. But due to the assumption $k_0 \geq \lceil \ell/2 \rceil$, this bound can be further specified:

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})} \leq \frac{\ell}{k_0} = \frac{\lceil \frac{\ell}{2} \rceil + \lfloor \frac{\ell}{2} \rfloor}{k_0} \leq \frac{\lceil \frac{\ell}{2} \rceil + \lceil \frac{\ell}{2} \rceil}{k_0} \leq \frac{2k_0}{k_0} = 2$$

□

The proof of the lemma shown above differentiates between two cases $k_0 < \lceil \ell/2 \rceil$ and $k_0 \geq \lceil \ell/2 \rceil$. To interpret these cases, the former case corresponds to a case

where an optimal (non-permutation) schedule σ^* processes a high number of jobs, whereas there latter case can be said to apply when σ^* is a "small" schedule where a small number of jobs is processed. Since it is possible that the number ℓ of processed jobs in σ^* is equal to the number n of the available jobs in the instance, i.e., $\ell = n$, it can also be said that these two cases correspond to "larger" and "smaller" instances, depending on how large n is in comparison to k_0. By replacing k_0 with the appropriate values for each $\mathsf{bufType} \in \{\mathit{intermediateBuffer}, \mathit{spanningBuffer}\}$, the following upper bounds for the solution quality ratio with respect to $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ are obtained from Lemma 5.1.13.

Theorem 5.1.10. *Let $\mathcal{I}$ be an instance of the Two-Stage VRP with Profits and Buffers containing n jobs with $\mathsf{bufType} = \mathit{intermediateBuffer}$ where $d_{ij} = 0$ holds for all edges e_{ij}. If $n > 6$, the solution quality ratio $R(\sigma^*, \mathcal{I}) / R(\sigma^*_{perm}, \mathcal{I})$ for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ is bounded from above by*

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}\, \mathcal{I})} \leq 1 + \frac{\lceil n/2 \rceil}{3}.$$

If $n \leq 6$, the ratio is bounded from above by 2.

Theorem 5.1.11. *Let $\mathcal{I}$ be an instance of the Two-Stage VRP with Profits and Buffers containing n jobs with $\mathsf{bufType} = \mathit{spanningBuffer}$ where $d_{ij} = 0$ holds for all edges e_{ij}. If $n > 12$, the solution quality ratio $R(\sigma^*, \mathcal{I}) / R(\sigma^*_{perm}, \mathcal{I})$ for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ is bounded from above by*

$$\frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}\, \mathcal{I})} \leq 1 + \frac{\lceil n/2 \rceil}{6}.$$

If $n \leq 12$, the ratio is bounded from above by 2.

5.1.5 Overview of Theoretical Results

The results for the Two-Stage VRP with Profits and Buffers and its restricted cases presented in the previous sections as well as the results from the theoretical analyses done in the previous chapters (see Section 3.2 and Section 4.2) can be visualized in a table. This gives a systematic overview on how the different special cases of the Two-Stage VRP with Profits and Buffers (referred to as "subclasses" in the following)

are related and can provide new insights on problems with a similar properties. In order to provide a succinct overview of the results, some notation is introduced.

- $\mathcal{D}^{R\geq Q}_{C_{\max}\leq B}$ is the decision problem whether for an instance of the Two-Stage VRP with Profits and Buffers (or an instance belonging to a subclass of that problem) and two positive numbers B, Q a solution σ exists with $C_{\max}(\sigma) \leq B$ and $R(\sigma) \geq Q$.

- $\mathcal{D}_{C_{\max}\leq B}$ is the decision problem whether for an instance of the Two-Stage VRP with Profits and Buffers (or an instance belonging to a subclass of that problem) and a positive number B a solution σ exists with $C_{\max}(\sigma) \leq B$ where all jobs are processed. This decision problem is based on the Two-Machine Flow Shop Problem with Buffers investigated in Chapter 3 where all jobs need to be processed (i.e., a solution σ needs to satisfy $R(\sigma) \geq \sum r_J$).

- The results presented in the following for the decision problems $\mathcal{D}_{C_{\max}\leq B}$ and $\mathcal{D}^{R\geq Q}_{C_{\max}\leq B}$ apply to both the case where only permutation schedules are allowed and the case where non-permutation schedules are allowed. The proofs for the latter case are straightforward to obtain from the former case by slightly modifying some arguments in the proofs.

- For an instance $\mathcal{I}$ of a Two-Stage VRP with Profits and Buffers, $\varphi_{\min C_{\max},\, R\geq Q}(\mathcal{I})$ is the solution quality ratio between the best possible permutation schedule σ^*_{perm} and an optimal schedule σ^* (that can be a non-permutation schedule) for the optimization problem $\mathcal{O}_{\min C_{\max},\, R\geq Q}$. Formally, it is calculated as

$$\varphi_{\min C_{\max},\, R\geq Q}(\mathcal{I}) = \frac{C_{\max}(\sigma^*_{perm}, \mathcal{I})}{C_{\max}(\sigma^*, \mathcal{I})},$$

  where $C_{\max}(\sigma, \mathcal{I})$ is the makespan (or "total length") of the solution σ for instance $\mathcal{I}$. Similarly, $\varphi_{\max R,\, C_{\max}\leq B}(\mathcal{I})$ is the solution quality ratio for the optimization problem $\mathcal{O}_{\max R,\, C_{\max}\leq B}$, i.e.,

$$\varphi_{\max R,\, C_{\max}\leq B}(\mathcal{I}) = \frac{R(\sigma^*, \mathcal{I})}{R(\sigma^*_{perm}, \mathcal{I})}.$$

- For a class $\mathcal{C}$ of problem instances belonging to the Two-Stage VRP with Profits and Buffers, an interesting question is how large the ratios $\varphi_{\min C_{\max},\, R\geq Q}(\mathcal{I})$

and $\varphi_{\max R,\, C_{\max} \leq B}(\mathcal{I})$ can potentially become. These values describe the potential "gap" between permutation and non-permutation schedules. For this reason, the notation $\Phi_{\min C_{\max},\, R \geq Q}$ and $\Phi_{\max R,\, C_{\max} \leq B}$ is used to describe upper bounds for $\varphi_{\min C_{\max},\, R \geq Q}(\mathcal{I})$ and $\varphi_{\max R,\, C_{\max} \leq B}(\mathcal{I})$, respectively, such that for all instances $\mathcal{I} \in \mathcal{C}$ belonging to $\mathcal{C}$ it holds that $\varphi_{\max R,\, C_{\max} \leq B}(\mathcal{I}) \leq \Phi_{\min C_{\max},\, R \geq Q}$ and $\varphi_{\max R,\, C_{\max} \leq B}(\mathcal{I}) \leq \Phi_{\max R,\, C_{\max} \leq B}$.

Using this notation, an overview of theoretical results obtained in this thesis is shown in Table 5.1. Every row describes a class of problems belonging to the Two-Stage VRP with Profits and Buffers, with its properties being described in the second to seventh column. The eighth column (abbreviated as "A" in Table 5.1) describes the complexity of the decision problem $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$. The ninth column (abbreviated as "B") shows the computational complexity of the decision problem $\mathcal{D}_{C_{\max} \leq B}$. In the tenth column (abbreviated as "C"), bounds for $\Phi_{\min C_{\max},\, R \geq Q}$ with respect to the considered class of problems are shown. Similarly, the eleventh column ("D") shows bounds for the upper bound $\Phi_{\max R,\, C_{\max} \leq B}$.

An example on how to read the table is given in the following. Consider row 8 of Table 5.1 which considers a subclass of Two-Stage VRP with Profits and Buffers where it is not assumed that $a_J = 0$ or $c = 0$ holds for all jobs. It is also not required that the processing time c on M_2 has the property "$a_J \leq c$ for all J" or "$a_J \geq c$ for all J". However, it is assumed that all travel lengths d_{ij} are zero for all edges. When this class of problem is considered, the decision problem $\mathcal{D}_{C_{\max} \leq B}$ is *NP*-complete which was shown in Theorem 3.2.1. The *NP*-completeness for $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$ follows from table cell B8, which is aforementioned *NP*-completeness for $\mathcal{D}_{C_{\max} \leq B}$. In addition, the results in this thesis provide the bounds $\frac{7}{6} \leq \Phi_{\min C_{\max},\, R \geq Q} \leq 2$ (shown in Corollary 3.2.3 and Theorem 3.2.8) for the upper bound $\Phi_{\min C_{\max},\, R \geq Q}$ and $\frac{4}{3} \leq \Phi_{\max R,\, C_{\max} \leq B} \leq 1 + \frac{\lceil n/2 \rceil}{3}$ (shown in Lemma 5.1.9 and Theorem 5.1.10) for the upper bound $\Phi_{\max R,\, C_{\max} \leq B}$. Note that these bounds are not sharp.

The notation "$\Phi = 1$" in Table 5.1 means that the upper bound Φ for the solution ratio φ between best possible permutation schedules and optimal schedules is exactly 1. Since it is not possible for the ratio φ on any instance to be smaller than 1, $\varphi = 1$ follows for all instances of the considered class of problems. In short, this means that for all problems of that class the set of optimal schedules always contains a permutation schedule.

This table contains all possible configurations for the second to seventh column using the values and restrictions considered in this thesis, but note that some configurations for problem classes are not displayed in the table since they coincide with other rows. For example, in rows 31–40 it is assumed that $c = 0$ holds from which $a_J \geq c$ trivially follows for all J. If $c = 0$ were to hold with the additional condition $a_J \leq c$, this would imply $a_J = 0$ for all J, but this case already corresponds to rows 47–52 in Table 5.1. Similarly, in rows 41–44 where $a_J = 0$ is assumed for all J, the case where $s_J = a_J$ holds for all J is not shown since this is equivalent to the case without buffer restrictions (rows 45 and 46).

Table 5.1: Overview of the theoretical properties for the Two-Stage VRP with Profits and Buffers and its subcases regarding computational complexity and potential gaps between permutation schedules and non-permutation schedules. The second and third column show whether it is assumed that all a_J are zero for all J or that the constant c is zero. The column "$a_J \gtreqless c$?" specifies whether $a_J \leq c$ for all J or $a_J \geq c$ for all J (or neither, indicated by "-") is assumed. The buffer is specified in the columns bType and bUsage (abbreviations for bufType and bufUsage, respectively), where bufType ="-" in combination with $s_J = 0$ refers to the case without buffer restrictions (i.e., infinite buffer). Spanning buffer and intermediate buffer are abbreviated as "span" and "inter", respectively. Column "$d_{ij} = 0$" shows whether $d_{ij} = 0$ is assumed for all edges e_{ij}. The eighth and ninth column show the complexity of the decision problems $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$ and $\mathcal{D}_{C_{\max} \leq B}$, with NPc being an abbreviation for "NP-complete". The tenth and eleventh column show bounds for the upper bound Φ for the ratio φ with respect to the optimization problems $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ and $\mathcal{O}_{\max R,\, C_{\max} \leq B}$. The parentheses written after a result indicate the table cells or theoretical properties shown in this thesis from which this result is directly derived. The running number in the first column and the notation A, B, C, D is used to refer to specific table cells more easily. Co, L and T are abbreviations for "Corollary", "Lemma" and "Theorem", respectively.

							A	B	C		D	
№	$a_J = 0$	$c = 0$	$a_J \gtreqless c$?	bType	bUsage	$d_{ij} = 0$	$\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$	$\mathcal{D}_{C_{\max} \leq B}$	$\Phi_{\min C_{\max},\, R \geq Q}$		$\Phi_{\max R,\, C_{\max} \leq B}$	
1	-	-	-	*span*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
2	-	-	-	*span*	$s_J = 1$	✓	NPc (B2)	NPc (T 3.2.1)	$\Phi = 1$	(T 3.2.3)	$\Phi = 1$	(L 3.2.6, L 5.1.1)
3	-	-	-	*span*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
4	-	-	-	*span*	$s_J = a_J$	✓	NPc (B4)	NPc (T 3.2.1)	$19/18 \leq \Phi \leq 2$	(Co 3.2.2, T 3.2.8)	$7/6 \leq \Phi \leq 1 + \lceil n/2 \rceil / 6$	(L 5.1.10, T 5.1.11)
5	-	-	-	*inter*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
6	-	-	-	*inter*	$s_J = 1$	✓	NPc (B6)	NPc (T 3.2.1)	$\Phi = 1$	(T 3.2.3)	$\Phi = 1$	(L 3.2.6, L 5.1.1)
7	-	-	-	*inter*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$7/6 \leq \Phi \leq 2$	(Co 3.2.3, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
8	-	-	-	*inter*	$s_J = a_J$	✓	NPc (B8)	NPc (T 3.2.1)	$7/6 \leq \Phi \leq 2$	(Co 3.2.3, T 3.2.8)	$4/3 \leq \Phi \leq 1 + \lceil n/2 \rceil / 3$	(L 5.1.9, T 5.1.10)
9	-	-	-	-	$s_J = 0$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
10	-	-	-	-	$s_J = 0$	✓	NPc (A40)	P [73]	$\Phi = 1$	[73]	$\Phi = 1$	([73], L 5.1.1)
11	-	-	$a_J \leq c$	*span*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
12	-	-	$a_J \leq c$	*span*	$s_J = 1$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
13	-	-	$a_J \leq c$	*span*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
14	-	-	$a_J \leq c$	*span*	$s_J = a_J$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
15	-	-	$a_J \leq c$	*inter*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
16	-	-	$a_J \leq c$	*inter*	$s_J = 1$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
17	-	-	$a_J \leq c$	*inter*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
18	-	-	$a_J \leq c$	*inter*	$s_J = a_J$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
19	-	-	$a_J \leq c$	-	$s_J = 0$	-	NPc (A51)	NPc (B51)	$10/9 \leq \Phi \leq 2$	(Co 5.1.2, T 5.1.4)	$3/2 \leq \Phi \leq n/2$	(T 5.1.5, T 5.1.6)
20	-	-	$a_J \leq c$	-	$s_J = 0$	✓	NPc (A40)	P (B10)	$\Phi = 1$	(C10)	$\Phi = 1$	(D10)

							A	B	C		D	
№	$a_J = 0$	$c = 0$	$a_J \gtreqless c$?	bType	bUsage	$d_{ij} = 0$	$\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$	$\mathcal{D}_{C_{\max} \leq B}$	$\Phi_{\min C_{\max},\, R \geq Q}$		$\Phi_{\max R,\, C_{\max} \leq B}$	
21	-	-	$a_J \geq c$	*span*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
22	-	-	$a_J \geq c$	*span*	$s_J = 1$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
23	-	-	$a_J \geq c$	*span*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
24	-	-	$a_J \geq c$	*span*	$s_J = a_J$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
25	-	-	$a_J \geq c$	*inter*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
26	-	-	$a_J \geq c$	*inter*	$s_J = 1$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
27	-	-	$a_J \geq c$	*inter*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
28	-	-	$a_J \geq c$	*inter*	$s_J = a_J$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
29	-	-	$a_J \geq c$	-	$s_J = 0$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
30	-	-	$a_J \geq c$	-	$s_J = 0$	✓	NPc (A40)	P (B10)	$\Phi = 1$	(C10)	$\Phi = 1$	(D10)
31	-	✓	$a_J \geq c$	*span*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
32	-	✓	$a_J \geq c$	*span*	$s_J = 1$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
33	-	✓	$a_J \geq c$	*span*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
34	-	✓	$a_J \geq c$	*span*	$s_J = a_J$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
35	-	✓	$a_J \geq c$	*inter*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
36	-	✓	$a_J \geq c$	*inter*	$s_J = 1$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
37	-	✓	$a_J \geq c$	*inter*	$s_J = a_J$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
38	-	✓	$a_J \geq c$	*inter*	$s_J = a_J$	✓	NPc (A40)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
39	-	✓	$a_J \geq c$	-	$s_J = 0$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
40	-	✓	$a_J \geq c$	-	$s_J = 0$	✓	NPc (T 5.1.8)	P (A10)	$\Phi = 1$	(C10)	$\Phi = 1$	(D10)
41	✓	-	$a_J \leq c$	*span*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.8)	$\Phi = 1$	(L 5.1.8, L 5.1.1)
42	✓	-	$a_J \leq c$	*span*	$s_J = 1$	✓	P (T 5.1.9)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
43	✓	-	$a_J \leq c$	*inter*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.8)	$\Phi = 1$	(L 5.1.8, L 5.1.1)
44	✓	-	$a_J \leq c$	*inter*	$s_J = 1$	✓	P (T 5.1.9)	P (T 3.2.2)	$\Phi = 1$	(T 3.2.2)	$\Phi = 1$	(T 3.2.2, L 5.1.1)
45	✓	-	$a_J \leq c$	-	$s_J = 0$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.8)	$\Phi = 1$	(L 5.1.8, L 5.1.1)
46	✓	-	$a_J \leq c$	-	$s_J = 0$	✓	P (T 5.1.9)	P (A10)	$\Phi = 1$	(C10)	$\Phi = 1$	(D10)
47	✓	✓	-	*span*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
48	✓	✓	-	*span*	$s_J = 1$	✓	P (L 5.1.6)	P (L 5.1.6)	$\Phi = 1$	(L 5.1.6)	$\Phi = 1$	(L 5.1.6)
49	✓	✓	-	*inter*	$s_J = 1$	-	NPc (A51)	NPc (B51)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
50	✓	✓	-	*inter*	$s_J = 1$	✓	P (L 5.1.6)	P (L 5.1.6)	$\Phi = 1$	(L 5.1.6)	$\Phi = 1$	(L 5.1.6)
51	✓	✓	-	-	$s_J = 0$	-	NPc (T 4.2.2)	NPc (T 5.1.7)	$\Phi = 1$	(L 5.1.7)	$\Phi = 1$	(L 5.1.7, L 5.1.1)
52	✓	✓	-	-	$s_J = 0$	✓	P (L 5.1.6)	P (L 5.1.6)	$\Phi = 1$	(L 5.1.6)	$\Phi = 1$	(L 5.1.6)

From Table 5.1, some notable observations can be made. For higher row numbers, the problems tend to become "easier" for both decision problems $\mathcal{D}_{C_{\max} \leq B}$ and $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$ as more restrictions are added to the problem. However, there are more cases where $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$ is *NP*-complete, whereas the corresponding problem for $\mathcal{D}_{C_{\max} \leq B}$ is still efficiently solvable. This is mainly due to the minimum score constraint where schedules σ additionally need to satisfy $R(\sigma) \geq Q$. It can also be seen that having non-zero travel times d_{ij} (in the seventh column) makes all of these problems *NP*-complete. Listing the complexity results using the columns $\mathcal{D}_{C_{\max} \leq B}$ and $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$ in Table 5.1 provides an overview of the "boundary" between efficiently solvable cases and "hard" cases of the Two-Stage VRP with Profits and Buffers.

It is not hard to see that the decision problem $\mathcal{D}_{C_{\max} \leq B}$ can be reduced to $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$ by choosing Q in the minimum profit constraint $R(\sigma) \geq Q$ to be so large that the profits of all available jobs need to be collected. From this it follows for any row in Table 5.1 that the *NP*-completeness for $\mathcal{D}_{C_{\max} \leq B}$ implies the *NP*-completeness for $\mathcal{D}^{R \geq Q}_{C_{\max} \leq B}$, which is another way to show some of the complexity results for $\mathcal{D}_{C_{\max} \leq B}$ (column `A`).

It is interesting to note how some of the results in the table are closely connected to well-known optimization problems. For example, `A40` is highly similar to the decision variant of the Knapsack problem, whereas `A51` and `B51` can be said to correspond to the decision variant of the Orienteering Problem and Traveling Salesperson Problem, respectively (see the corresponding *NP*-completeness proofs in Theorem 4.2.2 and Theorem 5.1.7). The Two-Machine Flow Shop with Buffers investigated in Chapter 3 corresponds to the rows 2,4,6 and 8 with the corresponding decision problems $\mathcal{D}_{C_{\max} \leq B}$ and the ratios $\Phi_{\min C_{\max},\, R \geq Q}$. Another example are the efficiently solvable subcases of the Two-Machine Flow Shop where the processing time on one machine dominate the times on the other machine (see Section 3.2.1): These correspond to several of the rows 11–28, in particular the decision problem $\mathcal{D}_{C_{\max} \leq B}$ for the cases with buffer restrictions where $d_{ij} = 0$ holds for all edges.

In general, it can be seen that for the majority of problems the existence of permutation schedules in the set of optimal schedules is guaranteed, but for decreasing row numbers where the problem classes become harder and more general, a "gap" arises between permutation schedules and non-permutation schedules. For the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ the gaps presented in this thesis tend to be larger

than for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ as some of them depend on the size of the instance, but it is possible that tighter bounds might be established in future research works as these bounds are not sharp.

5.2 A Metaheuristic Framework for the Two-Stage VRP with Profits and Buffers

The Two-Stage VRP with Profits and Buffers in its general formulation is a type of Vehicle Routing Problem that incorporates profits and the processing of jobs in two stages with a buffer constraint that has not yet been considered in the literature. It encompasses a variety of other combinatorial optimization problems, many of which are known or were shown in this thesis to be *NP*-hard. Due to this, formulating a heuristic algorithm that performs well poses a difficult problem.

For this reason, a framework for metaheuristic algorithms is presented in this section from which a variety of methods for the Two-Stage VRP with Profits and Buffers can be derived. In particular, this framework combines and generalizes aspects of the algorithms 2BF-ILS (presented for the Two-Machine Flow Shop with Buffers in Section 3.4) and VNS_{OP} (described for the Orienteering Problem in Section 4.3) since the optimization problems for which these algorithms were developed for are special cases of the Two-Stage VRP with Profits and Buffers. This shows how these algorithms are related and how they can be extended to tackle more general cases of the problem.

Before describing the framework, some remarks regarding the structure of the Two-Stage VRP with Profits and Buffers need to be made and some notation needs to be introduced. First, note that if it is assumed that in a schedule σ all jobs are processed as early as possible, it suffices to specify the permutations $\pi^1(\sigma)$, $\pi^2(\sigma)$ (or only $\pi(\sigma)$ if σ is a permutation schedule) in order to fully specify σ, including all starting times and completion times for the processed jobs and the paths traversed by M_1 and M_2 in the graph.

Under this assumption, elements σ of the solution space for the Two-Stage VRP with Profits and Buffers can be described by permutations of jobs. However, the permutations do not have to contain all jobs $\mathcal{J}$ in the graph, i.e., $\mathit{ProcessedJobs}(\sigma) \subseteq \mathcal{J}$, which means that a schedule only processes a subset of all available jobs. For this reason, it can be said that the solution space has a "hierarchical" structure: It is possible to define "subareas" $\mathcal{S}_{\mathcal{J}_0}$ of the solution space which are specified by a **subset of jobs** $\mathcal{J}_0 \subseteq \mathcal{J}$ and contain all solutions σ with $\mathit{ProcessedJobs}(\sigma) = \mathcal{J}_0$. By doing so the first "level" of the solution space is concerned with the set of processed jobs $\mathcal{J}_0$, whereas the second level further specifies the order in which the jobs in

$\mathcal{J}_0$ are processed in a solution, which due to the above assumption specifies the solution as a whole[2].

This structure is also reflected in the framework which consists of the following components:

- An initialization routine `Init` that performs a cursory search at a coarse level of the solution space. This is used to identify promising sets of jobs (i.e., a promising subarea $\mathcal{S}_{\mathcal{J}_0}$).
- Neighborhood functions $N, \mathcal{N}$: They can be defined for the different levels of the solution space. For example, on the lowest level for a given solution σ, $N(\sigma)$ can be defined to be the set of solutions that are "similar" to σ. On the level of subsets $\mathcal{J}_0$ it is possible for different neighborhood function $\mathcal{N}$ to assign a set $\mathcal{N}(\mathcal{J}_0)$ containing sets of jobs to $\mathcal{J}_0$ (i.e., $\mathcal{N}(\mathcal{J}_0) \subseteq \mathcal{P}(\mathcal{J})$, where $\mathcal{P}(\mathcal{J})$ is the power set of $\mathcal{J}$) where these subsets are "similar" to $\mathcal{J}_0$. However, how this "similarity" is defined needs to specified for each neighborhood function.
- Neighborhood evaluation functions f: These functions evaluate "neighbors", i.e., the elements contained in a neighborhood $N(\sigma)$ or $\mathcal{N}(\mathcal{J}_0)$ (which can be sets $\mathcal{J}' \subseteq \mathcal{J}$ of jobs or other schedules σ'). These functions are used to select elements of a neighborhood which are used by the optimization algorithm. In the case of evaluating subsets, it can also be said that these functions determine the movement between the aforementioned "subareas" $\mathcal{S}_{\mathcal{J}_0}$ of the solution space.
- A preliminary optimization routine $\texttt{Opt}_{\texttt{pre}}$: In contrast to the two previous components, $\texttt{Opt}_{\texttt{pre}}$ is used *inside* a subarea $\mathcal{S}_{\mathcal{J}_0}$ (i.e., for a given set of jobs $\mathcal{J}_0$) to quickly discover promising solutions σ_0 inside $\mathcal{S}_{\mathcal{J}_0}$.
- A local search routine $\texttt{Opt}_{\texttt{local}}$: This component is used to further improve a given solution σ_0 by applying local search operations. From the perspective

[2]It can be even said that there is a third "layer" where after the job permutations it is further specified *when* each job in $\mathcal{J}_0$ is processed. This means that specifying a job permutation π in the second level does not immediately lead to a schedule, but to the set $\mathcal{S}_\pi$ of all schedules σ that process jobs in the specified job order π, but at potentially different times. However, due to the assumption that all jobs are processed as early as possible, unnecessary idle times are excluded so that the sets $\mathcal{S}_\pi$ are simplified to only contain a single schedule σ.

of the solution space, $\texttt{Opt}_{\texttt{local}}$ performs a more thorough search in a subarea $\mathcal{S}_{\mathcal{J}_0}$ by exploring around a given point σ_0 inside that area.

The reason why an additional local search routine is incorporated (in constrast to VNS_{OP} for the Orienteering Problem where such a component is not used) is that in the Two-Stage VRP with Profits and Buffers the jobs have processing times on both machines and take up buffer space so that the makespan of a schedule is heavily dependent on the order of processed jobs. By using local search routines, a stronger focus can be placed on optimizing the order in which jobs are processed.

Based on these components, the proposed framework is referred to as a **Framework for Iterative Search Algorithms with Variable Neighborhoods** (ISAVaN framework) in the following. How these components are specified in detail determines the optimization behavior of the algorithm with respect to the different levels of the solution space. The general formulation of components above makes it possible to incorporate additional knowledge about the problem's structure into the algorithm, for example, by choosing the evaluation functions f such that the choice of new subareas $\mathcal{S}_{\mathcal{J}_0}$ is based on criteria that have importance for a given practical application.

In this thesis, the components are chosen such that they incorporate aspects of both 2BF-ILS and VNS_{OP}, two algorithms that obtained good results for two subcases of the Two-Stage VRP with Profits and Buffers. In particular, the following options for the components are considered:

- For a given set of jobs $\mathcal{J}_0 \subseteq \mathcal{J}$, $\mathcal{N}_{\mathsf{add}}(\mathcal{J}_0)$ contains all sets $\mathcal{J}' \subseteq \mathcal{J}$ of jobs that can be obtained from $\mathcal{J}$ by adding a job $J \in \mathcal{J}$ not contained in $\mathcal{J}_0$. Similarly, $\mathcal{N}_{\mathsf{remove}}(\mathcal{J}_0)$ contains all sets $\mathcal{J}' \subseteq \mathcal{J}$ of jobs that are obtained from $\mathcal{J}$ by removing an element from $\mathcal{J}$.

  For a permutation schedule σ_0, neighborhood $N_{\mathsf{add}}(\sigma_0)$ contains all schedules σ' that can be derived from σ_0 by adding a job $J \in \mathcal{J} \setminus \mathit{ProcessedJobs}(\sigma_0)$ to $\pi(\sigma_0)$. Conversely, $N_{\mathsf{remove}}(\sigma_0)$ is the set of all schedules σ' derived from σ_0 by removing a job $J \in \mathit{ProcessedJobs}(\sigma_0)$ from $\pi(\sigma)$. Note that these neighborhood functions assign sets of schedules σ' to σ_0 that belong to a different subareas $\mathcal{S}_{\mathcal{N}_{\mathsf{add}}(\mathit{ProcessedJobs}(\sigma_0))}$ and $\mathcal{S}_{\mathcal{N}_{\mathsf{remove}}(\mathit{ProcessedJobs}(\sigma_0))}$ of the solution space.

It is also possible to define "local" neighborhoods $N_1(\sigma_0)$ of solutions σ_0 where schedules $\sigma' \in N_1(\sigma_0)$ have the property $\mathit{ProcessedJobs}(\sigma') = \mathit{ProcessedJobs}(\sigma_0)$. Regarding this type of neighborhood, we consider N_i^{op} in the following with $op \in \{\mathit{insert}, \mathit{pairInsert}, \mathit{swap}\}$ and $i \in \{1, 2, \ldots, |\mathit{ProcessedJobs}(\sigma_0)|\}$. These neighborhood functions are similar to the neighborhoods used in 2BF-ILS (see Section 3.4) and assign sets $N_i^{op}(\sigma_0)$ of schedules σ' to σ_0, where σ' is obtained by applying one of the operations $op \in \{\mathit{insert}, \mathit{pairInsert}, \mathit{swap}\}$ with a given argument i to the permutation $\pi(\sigma_0)$.

For the neighborhoods containing schedules, it is assumed that they only contain feasible schedules, i.e., schedules that do not violate the buffer constraint. However, it is not required that they satisfy the budget constraint or the minimum score constraint since these constraints depend on which of the optimization problems $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ or $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ is considered.

- For a neighborhood $N(\sigma_0)$ with $N \in \{N_{\mathsf{add}}, N_{\mathsf{remove}}\}$ that contains schedules σ, the same evaluation functions $F = \{f_{\mathsf{length}}, f_{\mathsf{value}}, f_{\mathsf{ratio}}, f_{\mathsf{random}}\}$ as in algorithm $\mathrm{VNS}_{\mathrm{OP}}$ (see Section 4.3) are used to evaluate schedules $\sigma \in N(\sigma_0)$.

  For neighborhoods $\mathcal{N} \in \{\mathcal{N}_{\mathsf{add}}, \mathcal{N}_{\mathsf{remove}}\}$, a function $\tilde{f}_{\mathsf{value}}$ is used to evaluate a set $\mathcal{J}' \in \mathcal{N}(\mathcal{J}_0)$ as follows: $\tilde{f}_{\mathsf{value}}(\mathcal{J}', \mathcal{J}_0) = \sum_{J \in \mathcal{J}'} r_J - \sum_{J \in \mathcal{J}_0} r_J$. As the notation indicates, $\tilde{f}_{\mathsf{value}}$ and f_{value} are similar in that the evaluation only depends on the profits of the jobs, but not their length or the order in which they are processed. In fact, $\tilde{f}_{\mathsf{value}}$ is implicitly used when a new schedule is chosen with f_{value} since selecting a new schedule σ' from a neighborhood $N_{\mathsf{add}}(\sigma_0)$ using f_{value} corresponds to selecting the new set of jobs $\mathit{ProcessedJobs}(\sigma')$ from $\mathcal{N}_{\mathsf{add}}$ using $\tilde{f}_{\mathsf{value}}$ and randomly selecting a schedule σ' out of the schedules where the relative order of the jobs in $\mathit{ProcessedJobs}(\sigma_0) \cap \mathit{ProcessedJobs}(\sigma')$ is the same as in the original schedule σ_0.

- For a given permutation schedule σ_0, the following procedures are considered for the preliminary optimization routine $\mathtt{Opt}_{\mathtt{pre}}$:

  The value $\mathtt{Opt}_{\mathtt{pre}} = \mathit{NEH}$ indicates that the NEH heuristic is used to construct a new solution using the jobs in $\mathit{ProcessedJobs}(\sigma_0)$. This heuristic is originally proposed for Flow Shop problems [122], but it also performs well for Flow Shops with Buffers (see Section 3.3) and is straightforward to adapt to the Two-

Stage VRP with Profits and Buffers, since it only uses job insertions. When $\texttt{Opt}_{\texttt{pre}} = \textit{mNEH}$, the modified version mNEH is used, adapted to the Two-Stage VRP with Profits and Buffers. Furthermore, $\texttt{Opt}_{\texttt{pre}} = \textit{LK}$ means that the Chained Lin-Kernighan heuristic [10] (which was also used in VNS_{OP}) is called to quickly calculate a new solution based on the travel times d_{ij} between the nodes corresponding to the jobs in $\textit{ProcessedJobs}(\sigma_0)$. Finally, $\texttt{Opt}_{\texttt{pre}} = \textit{none}$ means that σ_0 is not modified and used as-is for the next step in the algorithm.

- Regarding the local search proutine $\texttt{Opt}_{\texttt{local}}$, the procedures used in the following are based on the local search routine used in 2BF-ILS, of which the relevant excerpt is shown in Algorithm 5.1. In this routine, local search operations are applied in an iterated manner until the resulting solution cannot be further improved using these operations. For a given permutation schedule σ_0, the following options are considered for $\texttt{Opt}_{\texttt{local}}$:

  If $\texttt{Opt}_{\texttt{local}} = \textit{LS}$, this routine is called with $op_1 = \textit{pairInsert}$, $op_2 = \textit{pairInsert}$, $op_3 = \textit{pairInsert}$, i.e., a sequence of length 3. This sequence is based on the one used in 2BF-ILS which was obtained by the algorithm configurator `irace` (see Section 3.5.3). For the value $\texttt{Opt}_{\texttt{local}} = \textit{sLS}$ ("short local search"), the local search routine in Algorithm 5.1 is called with one operation $op_1 = \textit{insert}$, which also obtained good results during the tuning with `irace`. In an extended variant of $\texttt{Opt}_{\texttt{local}} = \textit{LS}$, the value $\texttt{Opt}_{\texttt{local}} = \textit{LS}^*$ indicates that after the local search routine in Algorithm 5.1 is finished with the resulting schedule σ_1, additional pairwise swaps are performed with π^1 or π^2 (while keeping the other permutation constant) to construct non-permutation schedules. The reason for this choice is that the Two-Stage VRP with Profits and Buffers contains cases where the set of optimal schedules contains no permutation schedule (see the theoretical analysis in Section 5.1) so that it might be possible that an optimized permutation schedule can be further improved by testing similar non-permutation schedules. Finally, $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ means that no local search is performed so that the given solution σ_0 is used as-is for the next step in the algorithm.

The following analyses are restricted to aforementioned values for the components in the ISAVaN framework, but it is possible to consider additional neighborhoods, evaluation functions and optimization routines depending on the considered

Algorithm 5.1 Local Search subroutine of 2BF-ILS

Input: given solution σ_0, finite sequence of operations $(op_1, op_2, \ldots, op_s)$

$\sigma^{cur} \leftarrow \sigma_0$
$\pi^{rand} \leftarrow$ random job permutation of $ProcessedJobs(\sigma_0)$
for $j \in \{1, 2, \ldots, s\}$ **do**
    $op \leftarrow op_s$
    repeat ▷ local search
        for $k \in \{1, 2, \ldots, n\}$ **do**
            $i \leftarrow k$th element in π^{rand}
            $\hat{\sigma} \leftarrow$ solution in the neighborhood $N_i^{op}(\sigma^{cur})$ that minimizes $C_{\max}$
            if $C_{\max}(\hat{\sigma}) \leq C_{\max}(\sigma^{cur})$ **then**
                $\sigma^{cur} \leftarrow \hat{\sigma}$
            end if
        end for
    until σ^{cur} does not improve
end for
return σ^{cur}

problem and its structure. The initialization routine `Init` used in the following depends on the considered optimization problem ($\mathcal{O}_{\min C_{\max},\, R \geq Q}$ or $\mathcal{O}_{\max R,\, C_{\max} \leq B}$). Its details are outlined further below when the general structure of algorithms in this framework is presented.

Regarding the budget constraint $C_{\max}(\sigma) \leq B$ (for the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$) and the minimum score constraint $R(\sigma) \geq Q$ (for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$), it is interesting to note that adding and removing jobs (i.e., solution changes based on the neighborhoods N_{add} and N_{remove}) affects the validity of σ with respect to these constraints, but in opposing directions. In particular, removing the processing of jobs in σ increases the likelihood that the new solution σ' satisfies the budget constraint in the case of $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, but it becomes more likely that σ' does not satisfy the minimum score constraint when $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ is considered. The inverse holds for adding jobs to σ with respect to these constraints.

We say in the following that "the validity of σ changes", when a new solution σ' is selected from $N(\sigma)$ with $N \in \{N_{\mathsf{add}}, N_{\mathsf{remove}}\}$ where one of these solutions satisfies the budget constraint or minimum score constraint (depending on the considered optimization problem) and the other solution does not. This means that the new

Algorithm 5.2 General structure of an algorithm in the ISAVaN framework

1: $\sigma \leftarrow$ Result from initialization routine `Init`
2: **while** termination criterion not satisfied **do** ▷ main iterations
3: $f_1, f_2 \leftarrow$ random elements from the set F
4: **repeat**
5: $\sigma \leftarrow \arg\max_{\sigma' \in N_{\text{add}}(\sigma)} f_1(\sigma', \sigma)$
6: **until** σ changes its validity
7: Call preliminary optimization routine $\texttt{Opt}_{\texttt{pre}}$
8: Call local search routine $\texttt{Opt}_{\texttt{local}}$
9: **while** σ has not changed its validity **do**
10: $\sigma \leftarrow \arg\min_{\sigma' \in N_{\text{remove}}(\sigma)} f_2(\sigma', \sigma)$
11: **end while**
12: **end while**
13: **return** best solution found so far

solution σ' is satisfies that constraint (and the old solution σ does not) or that the new solution is violates that constraint (whereas σ does not).

Based on the remarks and the notation introduced above, the general form of an algorithm using the ISAVaN framework is shown in Algorithm 5.2. In this algorithm, the components described above are iteratively used to change $\mathit{ProcessedJobs}(\sigma)$ of a schedule σ, quickly calculate new, promising solutions and refine them using local search routines. Permitting validity changes in σ allows algorithms in this framework to also work with solutions which at first might violate a constraint, but can become valid and promising solutions after applying small modifications.

Note that the top half of Algorithm 5.2 uses a `repeat-until` loop, whereas a `while` loop is used at the bottom. This means that the latter loop might be skipped if the validity of σ has already changed during the preceding steps, since in $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ it might be more reasonable to add new jobs to a solution σ which already satisfies the budget constraint rather than remove jobs. For $\mathcal{O}_{\min C_{\max},\, R \geq Q}$, the validity of σ does not change for the any of the routines $\texttt{Opt}_{\texttt{pre}}$, $\texttt{Opt}_{\texttt{local}}$ used in this thesis so that the `while` loop is not skipped.

The initialization routines `Init` used in the following are outlined in Algorithm 5.3 and Algorithm 5.4. They perform a number k_{init} of initial iterations that have a similar structure to the iterations in the main algorithm, but with a stronger focus on exploration in order to do a cursory search of different subareas in the solution

Algorithm 5.3 Initialization routine `Init` used for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$

Input: initial iterations k_{init}

$\sigma \leftarrow$ empty solution (that only contains the depot node v_0)
for $it = 1, \ldots, k_{init}$ **do**
    repeat
        $\sigma \leftarrow \arg\min_{\sigma' \in N_{\mathsf{add}}(\sigma)} f_{\mathsf{random}}(\sigma', \sigma)$
    until $R(\sigma) \geq Q$
    Call preliminary optimization routine $\texttt{Opt}_{\texttt{pre}}$
    Call local search routine $\texttt{Opt}_{\texttt{local}}$
    while $R(\sigma) \geq Q$ **do**
        $\sigma \leftarrow \arg\min_{\sigma' \in N_{\mathsf{remove}}(\sigma)} f_{\mathsf{ratio}}(\sigma', \sigma)$
    end while
end for
return schedule σ

Algorithm 5.4 Initialization routine `Init` used for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$

Input: initial iterations k_{init}, initial insertion probability p

$\sigma \leftarrow$ empty solution (that only contains the depot node v_0)
for $it = 1, \ldots, k_{init}$ **do**
    for each node v not in σ **do**
        with probability p, insert v into σ at a random position
    end for
    Call preliminary optimization routine $\texttt{Opt}_{\texttt{pre}}$
    Call local search routine $\texttt{Opt}_{\texttt{local}}$
    while $C_{\max}(\sigma) > B$ **do**
        $\sigma \leftarrow \arg\min_{\sigma' \in N_{\mathsf{remove}}(\sigma)} f_{\mathsf{ratio}}(\sigma', \sigma)$
    end while
end for
return schedule σ

space. The initialization routine for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ has an additional parameter p for an insertion procedure that is similar to the one used in VNS_{OP} during the initial phase. For the experiments performed in the following sections with $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, the value $p = B / C_{\max}(\sigma_{LK})$ is chosen similar to VNS_{OP} where $C_{\max}(\sigma_{LK})$ is the total length of the schedule obtained after applying the Chained Lin-Kernighan heuristic $\texttt{Opt}_{\texttt{pre}} = LK$ on the set $\mathcal{J}$ of all jobs. However, this value slightly differs from the one used in VNS_{OP} (in particular, it contains no square root) since job processing times and buffer constraints need to be taken into account in the Two-Stage VRP with Profits and Buffers which make it more likely that a solution has a higher makespan that violates the budget constraint.

As for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$, the insertion procedure in Algorithm 5.3 is not performed with such a parameter, but until the minimum score constraint is satisfied. The reason for this is that it is not possible for a solution σ that violates the minimum score constraint to become valid by only reordering or removing jobs.

The general structure of the main iterations in Algorithm 5.2 is based on the Variable Neighborhood Search algorithm VNS_{OP}, as algorithms of this type have been successfully applied to other Vehicle Routing Problems (see the literature overviews given in Section 2.3 and Section 4.1). In fact, algorithm VNS_{OP} is an example for an algorithm in this framework for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ that can be obtained by choosing `Init` as in Algorithm 5.4 as well as setting $\texttt{Opt}_{\texttt{pre}} = LK$ and $\texttt{Opt}_{\texttt{local}} = noLS$.

Furthermore, an algorithm with iterations that are highly similar to 2BF-ILS is contained in this framework by setting $\texttt{Opt}_{\texttt{pre}} = mNEH$ and $\texttt{Opt}_{\texttt{local}} = LS$, with some differences being that 2BF-ILS does not incorporate the adding and removing of jobs (since the Two-Machine Flow Shop with Buffers only considers schedules where all jobs are processed, i.e., $\mathit{ProcessedJobs}(\sigma) = \mathcal{J}$) so that the mNEH algorithm is only called once and that 2BF-ILS uses a different perturbation mechanism. However, the framework in Algorithm 5.2 also incorporates perturbation and exploration mechanics using the initialization routine `Init` and randomized evaluation functions (line 3).

These are two examples for algorithms that can be modeled using the ISAVaN framework. However, a variety of other algorithms for the Two-Stage VRP with Profits and Buffers can be obtained by using other combinations for $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$, or even by adding new routines for these components, new neighbor-

hoods or other evaluation functions. The choice of these components characterizes the algorithm's behavior. This can be used to adjust whether the resulting algorithm focuses on exploring many different subareas $\mathcal{S}_{\mathcal{J}_0}$ of the solution space (i.e., many different sets of jobs for $\mathit{ProcessedJobs}(\sigma)$ by using various neighborhoods) or whether it focuses on thoroughly searching through a subarea of the solution space (for example, by having a stronger focus on local search routines $\texttt{Opt}_{\texttt{local}}$).

Furthermore, the general ideas of the framework and its components can be applied to other combinatorial optimization problems where their search spaces have a hierarchical structure that can be divided into "levels" or "layers". This can be done by properly adapting the components of the framework, in particular the neighborhood functions and the functions for evaluating neighbored solutions that characterize an algorithm's movement in the solution space.

5.3 Experimental Results

In the following sections, the ISAVaN framework and its components are evaluated in an experimental study. In order to do so, several algorithms derived from that framework are considered in the following. These algorithms use the initialization procedure, neighborhoods and the evaluation functions as described in the previous section. However, the components $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$ are varied by considering all combinations for $\texttt{Opt}_{\texttt{pre}} \in \{\mathit{NEH}, \mathit{mNEH}, \mathit{LK}, \mathit{none}\}$ and $\texttt{Opt}_{\texttt{local}} \in \{\mathit{LS}, \mathit{LS}^*, \mathit{sLS}, \mathit{noLS}\}$, resulting in 16 different algorithms from the ISAVaN framework. These algorithms can be considered to form a "spectrum" of different methods for the Two-Stage VRP with Profits and Buffers that also contains algorithms similar to 2BF-ILS and VNS_{OP}, the two heuristics developed in the previous chapters.

5.3.1 Problem Instances

The instances used for the experimental study were generated as follows. The 22 Orienteering Problem instances (see Section 4.4.3) were used as the basis by taking their graphs $G = (V, E)$, their travel time d_{ij} for edges $e_{ij} \in E$ and the node values r_v for $v \in V$ which are later used for the job profits. In order to generate instances for the Two-Stage VRP with Profits and Buffers a job J_v was defined for each node

$v \in V$ with job profit $r_{J_v} = r_v$ (recall that in contrast to the Orienteering Problem, the profits in this problem are assigned to jobs). The processing time $b_{J_v} = c$ on M_2 for all J_v was chosen to be the integer $c = \lfloor \overline{d_{ij}} \rfloor$, where $\overline{d_{ij}}$ is the average over all travel times d_{ij} for edges $e_{ij} \in E$. The processing time a_{J_v} for J_v on M_1 are randomly generated integers based on a uniform distribution over the set $\{1, 2, \ldots, 2 \cdot c\}$. The values are chosen in this way so that the makespan $C_{\max}(\sigma)$ is not overly determined by the processed jobs or the travel times d_{ij} in the graph.

Regarding the buffer, the amount s_{J_v} of buffer used by job J_v was set as $s_{J_v} = a_{J_v}$ for all J_v as these instances were observed to be harder than instances where $s_{J_v} = 1$ holds for all J_v (see Section 3.5). As for the buffer type, instances with intermediate buffer and instances with spanning buffer were generated. If bufType = *intermediateBuffer*, the buffer capacity Ω was chosen as $\Omega = q_{0.25}$ (with $q_{0.25}$ being the 25%-percentile over the processing times a_{J_v} on M_1) and for bufType = *spanningBuffer* the value $\Omega = \max_{J_v} s_{J_v} + q_{0.25}$ was chosen.

By doing so, the generated graphs and jobs contain characteristics of the hard orienteering problems and hard buffer flow shop problems (in particular, graphs from the Orienteering Problem benchmark `oplib` and buffer flow shop instances with $s_J = a_J$) as well as characteristics of road networks that can occur in practical applications (by using graphs based on the `city` instances from Section 4.4.3).

Based on these values, instances for both optimization problems $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ and $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ were generated as follows. For $\mathcal{O}_{\min C_{\max},\, R \geq Q}$, the minimum score constraint $R(\sigma) \geq Q$ was chosen with $Q = (\sum_{J_v} r_{J_v})/10$, so that a solution σ must have a total profit of at least Q in order to satisfy the minimum score constraint. For $\mathcal{O}_{\max R,\, C_{\max} \leq B}$, the available budget for the budget constraint $C_{\max}(\sigma) \leq B$ was chosen as $B = 2 \cdot B_{OP}$ where B_{OP} is the budget of the corresponding Orienteering Problem instance from which the instance of the Two-Stage VRP with Profits and Buffers is generated. The factor 2 was chosen since in contrast to the Orienteering Problem additional times for processing jobs and buffer constraints need to be taken into account. In total, 88 instances were generated using this method (44 for each optimization problem, based on the 22 OP instances combined with 2 buffer types).

5.3.2 Experimental Results for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$

Each of the algorithms ran on each instance with a time limit of 5, 10 and 15 minutes for small ($n \leq 50$), medium-sized ($50 < n \leq 100$) and large ($n > 100$) instances. The results were averaged over 30 repetitions. For the 42240 runs in total, a server cluster of Leipzig University with thirty-six 2.1-GHz-cores (each run being executed on one core) was used.

The evaluation methodology used in the following is similar to the methodology used in Section 3.5 and Section 4.4.5 for the Two-Machine Flow Shop with Buffers and the Orienteering Problem, respectively. In particular, progress curves (PC) and empirical cumulative distribution functions (ECDF) are used to evaluate the performance of an algorithm over its entire run time, with the "area under curve" (AUC) as an aggregate quality measure. These methods are similar to the ones used in a benchmarking study by Weise et al. [195].

Recall that progress curves show the quality of the best solution found so far over time, whereas the ECDF shows the percentage of runs over time that reach a target solution quality, with "solution quality" corresponding to $R(\sigma)$ for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ and $C_{\max}(\sigma)$ for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$. For the following experiments, the target solution quality on an instance I is chosen to be a quality value that deviates less than 5% from the best solution quality over all runs found on I. Figure 5.2 shows two example diagrams for progress curves and ECDF obtained from the numerical experiments.

Furthermore, the evaluation is performed with respect to the same time measures FE (function evaluations), SS (subsets) and NT (normalized time) as in the previous experiments performed in this thesis. For the normalized run time, the normalization factor used for an instance is the run time of the standard NEH heuristic applied on all jobs in an instance averaged over 30 repetitions. Using these time measures, the aforementioned diagrams were calculated for all instances, all algorithms and both optimization problems $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ and $\mathcal{O}_{\min C_{\max},\, R \geq Q}$.

First, the optimization problem $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ is considered, whereas the results for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ are presented in a separate section below due to their different nature. A particular focus is placed on separately analyzing the components $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$ in the following, but detailed results for all 16 algorithms can be found online at [94].

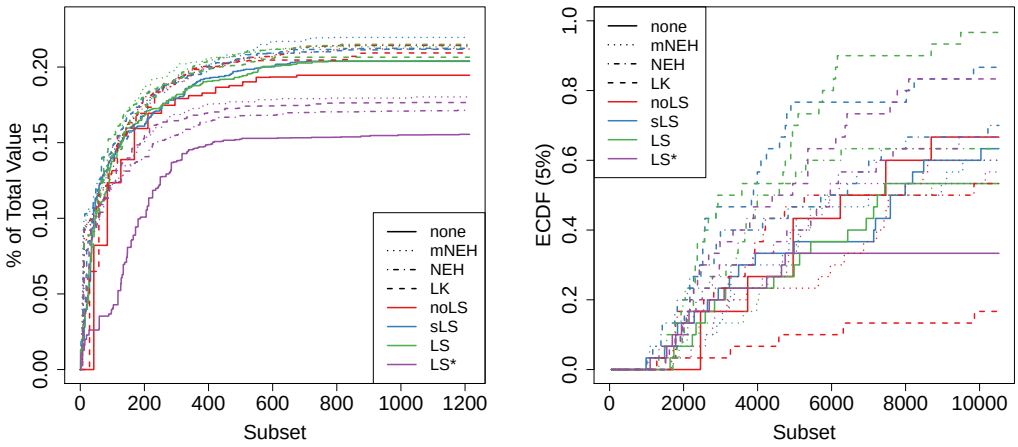


Figure 5.2: Example diagrams for progress curves and empirical cumulative distribution functions (ECDF). Left: Progress curves for the $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instance `LGF-brazil58-gen4-45.oplib` (containing $n = 58$ nodes and with spanning buffer). Right: Visualization of the ECDF for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Berlin-50-80000-1` (with $n = 50$ nodes and an intermediate buffer).

The Influence of $\texttt{Opt}_{\texttt{pre}}$ for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$

Recall the quality measure $AUC^{norm}_{I,A}$ (from Table 4.1 in Section 4.4.5) which describes for an algorithm A on instance I how "good" its AUC is in relation to the best performing algorithm A^* on that instance. Table 5.2 shows the $AUC^{norm}_{I,A}$ values for the PC curves (upper half) and ECDF curves (lower half), averaged over different sets of instances. The algorithms shown in this table use different optimization routines for $\texttt{Opt}_{\texttt{pre}}$, while $\texttt{Opt}_{\texttt{local}}$ is set to $\texttt{Opt}_{\texttt{local}} = noLS$ in order to minimize the influence of $\texttt{Opt}_{\texttt{local}}$.

It can be seen for $\texttt{tm} = FE$ and $\texttt{tm} = NT$ that the algorithm with $\texttt{Opt}_{\texttt{pre}} = LK$ obtains good results when the AUC of PC diagrams is considered. For $\texttt{tm} = SS$, the algorithms with $\texttt{Opt}_{\texttt{pre}} \in \{NEH, mNEH\}$ perform well which, with respect to that evaluation measure, means that these algorithms quickly improve their solutions over time. However, the values for ECDF are mixed and rather low (especially for $\texttt{tm} = SS$), which indicates that the target solution quality is hard to reach for these algorithms. A possible explanation for the low values for and $\texttt{tm} = SS$ is that

Table 5.2: Average $AUC_{I,A}^{norm}$ for different values of $\texttt{Opt}_{\texttt{pre}}$ with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ on $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances. A value close to 1 indicates that the average performance of an algorithm on an instance is similar to the best performance over all algorithms reached on that instance. The values are truncated to 3 decimal places and values in bold indicate the best average value (i.e., the value closest to 1) for each aggregation criterion and time measure.

PC ($\texttt{Opt}_{\texttt{pre}}$)	$\texttt{tm} = FE$				$\texttt{tm} = SS$				$\texttt{tm} = NT$			
	none	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*
Instance set												
`oplib`	0.928	**0.966**	0.962	0.992	0.804	**0.969**	0.966	0.939	0.928	0.967	0.963	**0.992**
`city`	0.933	0.941	0.944	**0.963**	0.840	0.968	**0.969**	0.922	0.929	0.936	0.94	**0.959**
Buffer type												
intermediate	0.927	0.949	0.949	**0.963**	0.824	0.972	**0.975**	0.923	0.923	0.945	0.946	**0.960**
spanning	0.937	0.941	0.945	**0.973**	0.843	**0.963**	0.962	0.927	0.935	0.939	0.942	**0.971**
All instances	0.932	0.945	0.947	**0.968**	0.834	**0.968**	**0.968**	0.925	0.929	0.942	0.944	**0.965**
ECDF ($\texttt{Opt}_{\texttt{pre}}$)	$\texttt{tm} = FE$				$\texttt{tm} = SS$				$\texttt{tm} = NT$			
	none	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*
Instance set												
`oplib`	0.000	**0.699**	0.637	0.647	0.000	**0.174**	0.080	0.135	0.000	**0.674**	0.610	0.650
`city`	0.091	**0.384**	0.411	0.377	0.085	**0.200**	0.165	0.102	0.087	0.360	**0.398**	0.390
Buffer type												
intermediate	0.074	**0.544**	0.516	0.429	0.071	**0.264**	0.150	0.097	0.064	**0.498**	0.490	0.437
spanning	0.074	0.337	0.389	**0.423**	0.069	0.126	**0.149**	0.120	0.078	0.336	0.382	**0.438**
All instances	0.074	0.441	**0.452**	0.426	0.070	**0.195**	0.150	0.108	0.071	0.417	0.436	**0.437**

many subsets need to be tested (i.e., a long time with respect to that time measure) before the target solution quality is reached, which is understandable since the target criterion $R(\sigma)$ directly depends on the considered subset of jobs.

It can also be observed that algorithms with $\texttt{Opt}_{\texttt{pre}} = \textit{none}$ do not perform well (especially with respect to ECDF diagrams) as they obtain the lowest values. In some cases is even 0 which means that the target solution quality was not reached in any of the runs. A possible explanation for this is that the resulting algorithm does not try to reduce the makespan of solutions violating the budget constraint so that potentially good solutions are missed.

Furthermore, it is interesting to note that for $\texttt{Opt}_{\texttt{pre}} = \textit{LK}$ and $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ the algorithm is highly similar to VNS_{OP} developed for the Orienteering Problem. The results show that for selected evaluation measures VNS_{OP} also performs fairly well

for the general Two-Stage VRP with Profits and Buffers. This can also be observed in the progress curves (see, e.g., Figure 5.3 left) where with $\texttt{Opt}_{\texttt{pre}} = LK$ a high solution quality is quickly reached.

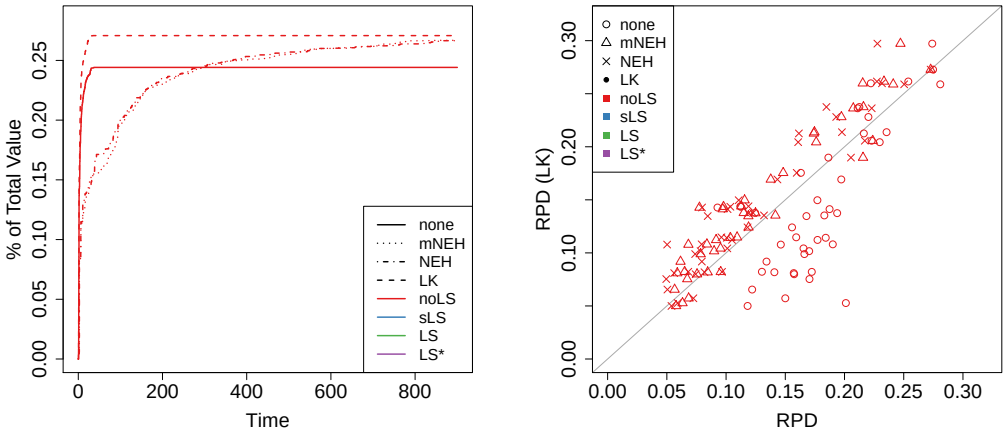


Figure 5.3: Left: Visualization of progress curves for the $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instance `LGF-gr120-gen4-85.oplib-590` (containing $n = 120$ nodes and using a spanning buffer) and the algorithms with $\texttt{Opt}_{\texttt{local}} = noLS$. Right: Scatter plot of average RPD values for the solution quality with $\texttt{Opt}_{\texttt{pre}} = LK$ at the end of the run in relation to the respective RPD values of algorithms with different $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}} = noLS$. The grey line marks the diagonal line $y = x$ such that points above (below) the line indicate that an algorithm obtained a better (worse) final solution than the algorithm with $\texttt{Opt}_{\texttt{pre}} = LK$.

However, the solution quality at the end of the run tends to be higher when $\texttt{Opt}_{\texttt{pre}} \in \{NEH, mNEH\}$ is used, as can be seen by plotting the RPD ("relative percentage difference") values of final solution quality for the different values of $\texttt{Opt}_{\texttt{pre}}$ against $\texttt{Opt}_{\texttt{pre}} = LK$ as shown in Figure 5.3 right. The RPD value is calculated as $RPD_{\texttt{Opt}_{\texttt{pre}}} = (R^* - R_{\texttt{Opt}_{\texttt{pre}}})/R^*$ (where $R_{\texttt{Opt}_{\texttt{pre}}}$ is the final solution quality of the algorithm on an instance with the corresponding value for $\texttt{Opt}_{\texttt{pre}}$ and R^* is the best solution quality found on the same instance over all runs) and the points are plotted with the coordinates (RPD_ω, RPD_{LK}) with $\omega \in \{none, mNEH, NEH\}$. The resulting scatterplot is consistent with the previous observation for ECDF diagrams in Table 5.2 that the target solution quality is reached more consistently when

$\texttt{Opt}_{\texttt{pre}} \in \{NEH, mNEH\}$. In addition, it can be seen in Figure 5.3 right that the algorithm with $\texttt{Opt}_{\texttt{pre}} = LK$ tends to obtain better results than the one using $\texttt{Opt}_{\texttt{pre}} = none$.

The Influence of $\texttt{Opt}_{\texttt{local}}$ for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$

Regarding the local search routine $\texttt{Opt}_{\texttt{local}}$, Table 5.3 shows the average $AUC^{norm}_{I,A}$ values for the algorithms with $\texttt{Opt}_{\texttt{pre}} = none$.

Table 5.3: Average $AUC^{norm}_{I,A}$ for different values of $\texttt{Opt}_{\texttt{local}}$ with $\texttt{Opt}_{\texttt{pre}} = none$ on $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances. Values close to 1 are preferable.

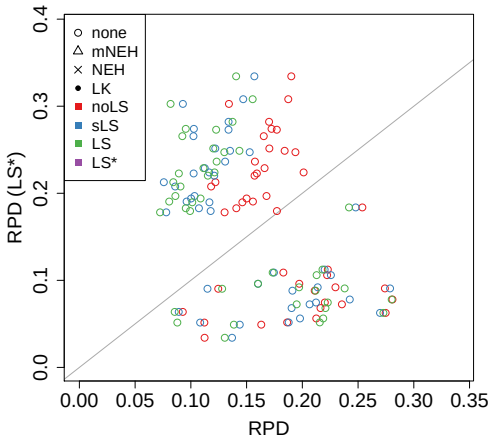
PC ($\texttt{Opt}_{\texttt{local}}$)	tm = *FE*				tm = *SS*				tm = *NT*			
	noLS	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**
Instance set												
oplib	0.928	**0.968**	0.967	0.786	0.804	**0.837**	0.835	0.648	0.928	**0.968**	0.967	0.795
city	0.933	**0.959**	0.956	0.892	0.840	**0.873**	**0.873**	0.779	0.929	**0.955**	0.952	0.895
Buffer type												
intermediate	0.927	**0.955**	0.953	0.885	0.824	**0.860**	0.858	0.784	0.923	**0.951**	0.949	0.889
spanning	0.937	**0.967**	0.964	0.860	0.843	**0.873**	**0.873**	0.727	0.935	**0.964**	0.961	0.865
All instances	0.932	**0.961**	0.958	0.872	0.834	**0.867**	0.866	0.755	0.929	**0.957**	0.955	0.877
ECDF ($\texttt{Opt}_{\texttt{local}}$)	tm = *FE*				tm = *SS*				tm = *NT*			
	noLS	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**
Instance set												
oplib	0.000	0.260	**0.421**	0.037	0.000	0.070	**0.105**	0.000	0.000	0.263	**0.433**	0.050
city	0.091	0.275	0.222	**0.416**	0.085	0.154	0.060	**0.207**	0.087	0.275	0.220	**0.429**
Buffer type												
intermediate	0.074	0.192	0.249	**0.374**	0.071	0.129	0.095	**0.254**	0.064	0.192	0.255	**0.391**
spanning	0.074	**0.353**	0.267	0.319	0.069	**0.148**	0.041	0.084	0.078	**0.354**	0.263	0.329
All instances	0.074	0.272	0.258	**0.347**	0.070	0.138	0.068	**0.169**	0.071	0.273	0.259	**0.360**

For progress curve diagrams (PC), the best results are obtained in all time measures when $\texttt{Opt}_{\texttt{local}} = sLS$. This indicates that a short local search routine allows the resulting algorithms to quickly improve their solutions so that a good optimization behavior over time is obtained. The results are also better than algorithms with $\texttt{Opt}_{\texttt{local}} = noLS$, likely due to job processing times and the buffer that, in contrast to the Orienteering Problem, make it more important to take the order of processed jobs into account.

Regarding ECDF diagrams, $\texttt{Opt}_{\texttt{local}} = sLS$ only leads to good results for the instances with spanning buffer, whereas in other cases the algorithm with $\texttt{Opt}_{\texttt{local}} = LS^*$ tends to perform better. This is likely because algorithms with LS^* try to reduce the length of invalid solutions more than other algorithms, potentially leading to more valid solutions where a high number of jobs are processed. Inspecting the solutions generated during the runs showed that, on average, approximately 3.6 non-permutation solutions are found per iteration that are as good as or better than the solution obtained after the local search routine. However, finding these solutions takes more time so that algorithms with $\texttt{Opt}_{\texttt{local}} = LS^*$ do not obtain the best results for PC diagrams. A similar observation, albeit at a smaller scale, can also be made for $\texttt{Opt}_{\texttt{local}} = LS$ on the `oplib` instances with respect to the ECDF diagrams. The values for that row in Table 5.3 indicate that reaching the target solution quality is harder on these instances than for the `city` instances.

Plotting the RPD values for the solution quality at the end of the runs (see Figure 5.4 left) shows that there are many instances where the algorithm with $\texttt{Opt}_{\texttt{local}} = LS^*$ obtains better results due to testing non-permutation solutions (the points above the diagonal line). However, there are also instances (points below the diagonal) where this is not the case, likely due to the additional time needed to test such solutions. Due to this, it cannot be said in general that $\texttt{Opt}_{\texttt{local}} = LS^*$ leads to better solutions at the end of the run as this also depends on other factors, such as the properties of the instance and the available computation time. This can also be seen when performing statistical tests using the two-sided sign test for paired data, which is a non-parametric test. The results (see Figure 5.4 right) show that no statistically significant difference can be seen for $\texttt{Opt}_{\texttt{local}} = LS^*$ when compared to the algorithms with other local search routines $\texttt{Opt}_{\texttt{local}}$. However, the tests show that algorithms with $\texttt{Opt}_{\texttt{local}} \in \{sLS, LS\}$ consistently obtain better solutions at the end of the run than with $\texttt{Opt}_{\texttt{local}} = noLS$.

To summarize the results, the appropriate choice of components $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$ depends on several factors. For example, if solutions of acceptable quality need to be reached quickly, then a feasible choice might be $\texttt{Opt}_{\texttt{pre}} = LK$ along with $\texttt{Opt}_{\texttt{local}} = sLS$, in which case many subsets of jobs are tested over a short time which is reasonable for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$. Also, these values lead to good results with respect to PC diagrams. But for certain types of instances and if enough computation time is available, then $\texttt{Opt}_{\texttt{pre}} \in \{mNEH, NEH\}$ with $\texttt{Opt}_{\texttt{local}} = LS^*$ might also be a feasible



	noLS	*sLS*	*LS*	*LS**
noLS		▲	▲	-
sLS	◀		-	-
LS	◀	-		-
*LS**	-	-	-	

Figure 5.4: Left: Left: Scatter plot of average RPD values for the solution quality with $\texttt{Opt}_{\texttt{local}} = LS^*$ at the end of the run in relation to the respective RPD of algorithms with different values for $\texttt{Opt}_{\texttt{local}}$ and $\texttt{Opt}_{\texttt{pre}} = \textit{none}$. Right: Results of the pairwise comparisons between different $\texttt{Opt}_{\texttt{local}}$ using the two-sided sign test over all $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instances ($n = 44$). A black triangle indicates that the measured difference is statistically significant ($p < 0.05/6$ due to Bonferroni correction) and that the algorithm at which the triangle is pointed at is significantly better according to the test statistic. Note that the tables are symmetric since the 3 possible pairwise comparisons for each of the three points in time were performed with two-sided tests.

choice where potentially better solutions can be found, even though the resulting algorithms need more time to optimize the solutions. However, the performance of these methods is comparable to the configuration $(\texttt{Opt}_{\texttt{pre}}, \texttt{Opt}_{\texttt{local}}) = (LK, noLS)$ which is similar to algorithm VNS_{OP} that obtained good results on the Orienteering Problem. This indicates that the other algorithms can achieve a performance similar to VNS_{OP}.

5.3.3 Experimental Results for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$

In this section, results are presented with respect to the 44 instances for the optimization problem $\mathcal{O}_{\min C_{\max},\, R \geq Q}$. Similar to the analysis in previous section, separate analyses are performed for the components $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$.

The Influence of $\texttt{Opt}_{\texttt{pre}}$ for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$

Table 5.4 shows the averaged $AUC^{norm}_{I,A}$ values for PC and ECDF diagrams regarding the algorithms with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ and different options for $\texttt{Opt}_{\texttt{pre}}$.

Table 5.4: Average $AUC^{norm}_{I,A}$ for different values of $\texttt{Opt}_{\texttt{pre}}$ with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ on $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances. A value close to 1 indicates that the average performance of an algorithm on an instance is similar to the best performance over all algorithms reached on that instance. The values are truncated to 3 decimal places and values in bold indicate the best average value (i.e., the value closest to 1) for each aggregation criterion and time measure.

PC ($\texttt{Opt}_{\texttt{pre}}$)	tm = *FE*				tm = *SS*				tm = *NT*			
	none	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*
Instance set												
oplib	**1.367**	1.474	1.606	1.884	2.301	**2.222**	2.237	2.527	1.372	**1.325**	1.329	1.870
city	2.160	2.058	2.112	**2.049**	2.480	1.937	**1.928**	2.214	2.188	**2.013**	2.032	2.070
Buffer type												
intermediate	2.130	2.118	2.170	**1.989**	2.453	**1.989**	1.994	2.290	2.145	2.038	2.036	**1.993**
spanning	1.902	**1.786**	1.871	2.049	2.442	1.988	**1.975**	2.251	1.934	**1.738**	1.772	2.074
All instances	2.016	**1.952**	2.020	2.019	2.448	1.989	**1.985**	2.271	2.040	**1.888**	1.904	2.034
ECDF ($\texttt{Opt}_{\texttt{pre}}$)	tm = *FE*				tm = *SS*				tm = *NT*			
	none	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*	*none*	*NEH*	*mNEH*	*LK*
Instance set												
oplib	0.375	0.342	0.341	**0.570**	0.304	0.269	0.266	**0.473**	0.375	0.351	0.350	**0.570**
city	0.437	**0.519**	0.497	0.511	0.372	**0.457**	0.423	0.446	0.436	**0.525**	0.506	0.516
Buffer type												
intermediate	**0.535**	0.427	0.480	0.509	0.423	0.372	0.391	**0.457**	**0.534**	0.439	0.497	0.518
spanning	0.316	**0.546**	0.457	0.535	0.296	**0.474**	0.398	0.445	0.316	**0.549**	0.459	0.533
All instances	0.425	0.487	0.468	**0.522**	0.360	0.423	0.395	**0.451**	0.425	0.494	0.478	**0.526**

The values for the progress curves (PC) show that the algorithms with $\texttt{Opt}_{\texttt{pre}} \in \{\textit{mNEH}, \textit{NEH}\}$ obtain good results fairly consistently. This can also be seen in the progress curves, e.g., the ones shown in Figure 5.5 left where the algorithms with these values for $\texttt{Opt}_{\texttt{pre}}$ find shorter solutions more quickly. A possible explanation for this is that both the mNEH and NEH heuristic test insertions of jobs into many positions so that they can find fairly short solutions; this is not done in the Lin-Kernighan heuristic $\texttt{Opt}_{\texttt{pre}} = \textit{LK}$ and especially not in the algorithm with $\texttt{Opt}_{\texttt{pre}} = \textit{none}$ which does not perform well in the shown example. Interestingly, for the Two-

Machine Flow Shop Problem with Buffers, which is a special case of $\mathcal{O}_{\min C_{\max},\, R \geq Q}$, these heuristics are also known to perform well (in particular, mNEH is also used in 2BF-ILS) which indicates they might also be promising heuristics for the general Two-Stage VRP with Profits and Buffers.

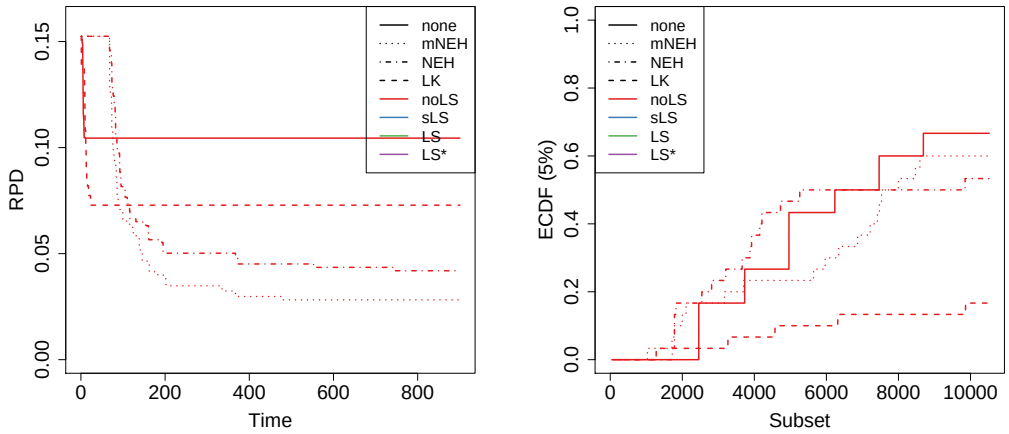


Figure 5.5: Left: Visualization of progress curves for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Leipzig-150-80000-2` (containing $n = 150$ nodes and with spanning buffer) and the algorithms with $\texttt{Opt}_{\texttt{local}} = noLS$. Right: ECDF curves for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Berlin-50-80000-1` (containing $n = 50$ nodes and with intermediate buffer) and for the algorithms with $\texttt{Opt}_{\texttt{local}} = noLS$.

Regarding the ECDF diagrams, the results are mixed as the closest values to 1 are not consistently obtained by any of the algorithms. When averaged over all instances, the values for $\texttt{Opt}_{\texttt{pre}} = LK$ are the ones closest to 1, but there are also cases where $\texttt{Opt}_{\texttt{pre}} = NEH$ obtains better results than with $\texttt{Opt}_{\texttt{pre}} = LK$ (for an example, see ECDF diagram shown in Figure 5.5 right). This indicates that both of these choices for $\texttt{Opt}_{\texttt{pre}}$ can be viable, depending on whether the focus of the resulting algorithm should be closer to the Flow Shop Problem (with $\texttt{Opt}_{\texttt{pre}} = NEH$) or to the Traveling Salesperson Problem (with the Chained Lin-Kernighan heuristic $\texttt{Opt}_{\texttt{pre}} = LK$).

In addition, statistical tests for comparing the four algorithms' average solution quality at the end of the run were performed using the two-sided sign test for paired

data which is a non-parametric test. However, no statistically significant differences were observed ($p \geq 0.05/6$ due to Bonferroni correction with $n = 44$). This can also be visualized by plotting the RPD values for different $\texttt{Opt}_{\texttt{pre}}$ in comparison to $\texttt{Opt}_{\texttt{pre}} = NEH$ in Figure 5.6 left where none of the other values for $\texttt{Opt}_{\texttt{pre}}$ consistently obtain better solutions (above the diagonal) or worse solutions (below the diagonal) than with $\texttt{Opt}_{\texttt{pre}} = NEH$. In fact, many points are very close to the diagonal, indicating that a similar performance at the end of the run is reached. These results indicate that the choice of $\texttt{Opt}_{\texttt{pre}}$ for this optimization problem rather affects an algorithms performance over the run time rather than the performance reached at the end of the run.

The Influence of $\texttt{Opt}_{\texttt{local}}$ for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$

Similar to the analysis in the previous section, Table 5.5 shows the averaged $AUC^{norm}_{I,A}$ values for the algorithms with varying $\texttt{Opt}_{\texttt{local}}$ while $\texttt{Opt}_{\texttt{pre}}$ is set to $\texttt{Opt}_{\texttt{pre}} = none$.

Table 5.5: Average $AUC^{norm}_{I,A}$ for different values of $\texttt{Opt}_{\texttt{local}}$ with $\texttt{Opt}_{\texttt{pre}} = none$ on $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances.

PC ($\texttt{Opt}_{\texttt{local}}$)	tm = *FE*				tm = *SS*				tm = *NT*			
	noLS	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**
Instance set												
oplib	**1.367**	1.859	2.362	2.155	2.301	2.273	2.229	**1.287**	**1.372**	1.874	2.398	2.069
city	**2.160**	2.526	2.940	2.974	2.480	2.074	**2.049**	2.203	**2.188**	2.565	2.983	3.002
Buffer type												
intermediate	**2.130**	2.363	2.955	2.997	2.453	2.065	2.039	**1.989**	**2.145**	2.397	2.996	2.982
spanning	**1.902**	2.447	2.714	2.654	2.442	2.156	2.124	**2.084**	**1.934**	2.482	2.757	2.682
All instances	**2.016**	2.405	2.835	2.825	2.448	2.110	2.081	**2.037**	**2.040**	2.440	2.877	2.832
ECDF ($\texttt{Opt}_{\texttt{local}}$)	tm = *FE*				tm = *SS*				tm = *NT*			
	noLS	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**	*noLS*	*sLS*	*LS*	*LS**
Instance set												
oplib	0.375	0.342	0.322	**0.593**	0.304	0.308	0.341	**0.652**	0.375	0.340	0.316	**0.577**
city	**0.437**	0.400	0.348	0.344	0.372	0.424	0.428	**0.447**	**0.436**	0.396	0.342	0.343
Buffer type												
intermediate	**0.535**	0.414	0.375	0.394	0.423	0.413	0.458	**0.495**	**0.534**	0.406	0.367	0.393
spanning	0.316	0.366	0.311	**0.383**	0.296	0.392	0.367	**0.474**	0.316	0.365	0.308	**0.378**
All instances	**0.425**	0.390	0.343	0.389	0.360	0.403	0.412	**0.485**	**0.425**	0.386	0.338	0.386

Interestingly, for tm $\in \{FE, NT\}$ the algorithm with $\texttt{Opt}_{\texttt{local}} = noLS$ performs the best, whereas for tm $= SS$ the best results on average are obtained when $\texttt{Opt}_{\texttt{local}} = LS^*$. This might be surprising since for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ the target criterion is the makespan $C_{\max}(\sigma)$ that is heavily influenced by the order in which jobs are processed. However, a possible explanation for this is that the makespan $C_{\max}(\sigma)$ of a solution σ also is also indirectly determined by the set of jobs $ProcessedJobs(\sigma)$ and thus, the subset of nodes that need to be traversed in order to process these jobs. This indicates that the problem of selecting a subset of nodes also plays an important for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$. By choosing $\texttt{Opt}_{\texttt{local}} = noLS$, the resulting algorithm focuses on trying many subsets in a short time, which can be a reasonable choice if many subsets are available that satisfy the minimum score constraint.

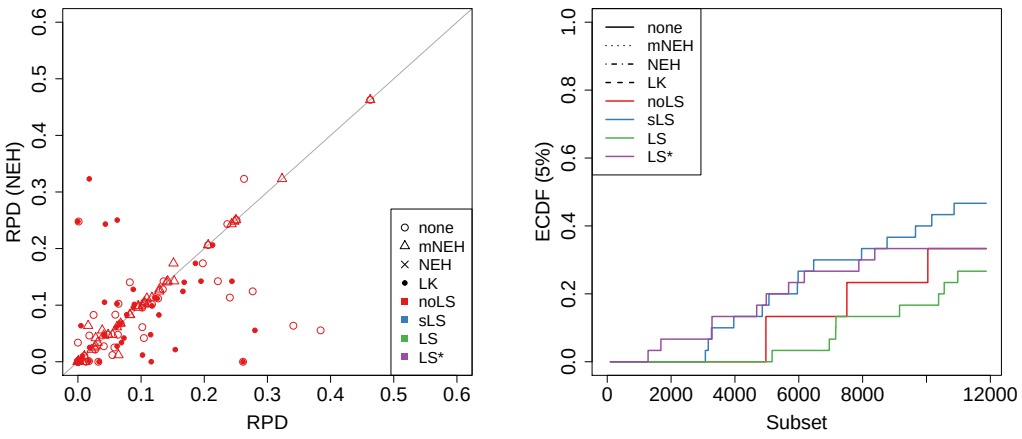


Figure 5.6: Left: Scatter plot of average RPD values for the solution quality with $\texttt{Opt}_{\texttt{pre}} = NEH$ at the end of the run in relation to the respective RPD values of algorithms with different $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}} = noLS$. The grey line marks the diagonal line $y = x$ such that points above (below) the line indicate that an algorithm obtained a better (worse) final solution than the algorithm with $\texttt{Opt}_{\texttt{pre}} = NEH$. Right: ECDF curves for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Berlin-100-80000-1` (containing $n = 100$ nodes and with intermediate buffer) for the algorithms with $\texttt{Opt}_{\texttt{pre}} = none$.

However, regarding the evaluation measure ECDF the results are rather mixed with both $\texttt{Opt}_{\texttt{local}} = noLS$ and $\texttt{Opt}_{\texttt{local}} = LS^*$ obtaining the values closest to one.

Especially for $\texttt{tm} = SS$, the algorithm with $\texttt{Opt}_{\texttt{local}} = LS^*$ outperforms the other algorithms in Table 5.5. Even though the ECDF curves do not reach particularly high values (see Figure 5.6 right for an example), the target solution quality is reached more consistently with $\texttt{Opt}_{\texttt{local}} = LS^*$ than with $\texttt{Opt}_{\texttt{local}} = noLS$ when time is measured in evaluated subsets.

A likely explanation for this is that the local search procedure $\texttt{Opt}_{\texttt{local}} = LS^*$ checks many permutations for a given job set $ProcessedJobs(\sigma)$. Indeed, inspecting the log data for the generated solutions shows that, on average, approximately 5.1 non-permutation solutions are found per iteration that are as good as or better than the previous solution found after the local search routine. However, with respect to the time measure $\texttt{tm} = SS$, testing all of these permutations counts as the "same" subset so that it is more likely that only a few subsets need to checked before finding a solution with short makespan (that reaches the target solution quality).

Thus, it can be said that $\texttt{Opt}_{\texttt{local}} = LS^*$ provides a feasible option when the focus is placed on not testing a large number of subsets. In the case where not much time is available or when the number of function evaluations should be kept low, the algorithm with $\texttt{Opt}_{\texttt{local}} = noLS$ is a suitable choice as it quickly tests different subareas of the solution space.

5.4 Summary

This chapter considered the Two-Stage Vehicle Routing Problem with Profits and Buffers which generalizes the Two-Machine Flow Shop Problem with Buffers and the Orienteering Problem investigated in Chapter 3 and Chapter 4, respectively. A theoretical analysis regarding computational complexity, existence of permutation schedules in the set of optimal schedules, and potential gaps in attainable solution quality between permutation schedules and non-permutation schedules was performed. These aspects were investigated for the general problem and a variety of special cases that arise when restrictions are added which are similar to the ones occurring in the Two-Machine Flow Shop with Buffers and the Orienteering Problem.

The theoretical results were arranged in a table to illustrate how various subclasses of the Two-Stage VRP with Profits and Buffers and their properties regarding the aforementioned aspects are related. This table also contains the theoretical results

obtained for the Two-Machine Flow Shop Problem with Buffers and the Orienteering Problem, so it can be considered to provide an overview of all theoretical results obtained in this thesis.

Since this problem encompasses a variety of other combinatorial optimization problems, the framework "ISAVaN" (for "Iterative Search Algorithms with Variable Neighborhoods") was proposed as a framework for metaheuristic algorithms that can be applied to the Two-Stage VRP with Profits and Buffers. It generalizes aspects of 2BF-ILS and VNS_{OP}, two algorithms that obtained good results for the Two-Machine Flow Shop with Buffers and the Orienteering Problem, but various other algorithms can be derived from this framework by changing the neighborhoods and optimization subroutines used by the algorithm. These components adjust the algorithm's focus in the solution space that, in the Two-Stage VRP with Profits and Buffers, has a layered structure. The general formulation of the framework also allows it to be applied to other combinatorial optimization problems by adapting the components, in particular the neighborhoods and the local search routines in accordance with the problem's structure.

In an experimental study, various algorithms derived from the ISAVaN framework for the Two-Stage VRP with Profits and Buffers were compared. The instances used for the experiments are based on characteristics found in difficult instances for the Two-Machine Flow Shop Problem with Buffers and the Orienteering Problem. The evaluation methodology is focused on the performance of algorithms over the entire run time with respect to several time measures using graphical evaluation measures, empirical cumulative distribution functions (ECDF) and the area under curve (AUC) as an aggregate quality measure that is averaged over different sets of instances. In addition, statistical tests to evaluate the significance of differences at selected points in time as well as scatter plots due to the paired nature of the measured data were used. As briefly mentioned in Section 2.4, these methods were also used in the experimental studies for the Two-Machine Flow Shop with Buffers and the Orienteering Problem, which illustrates that for a variety of problems which are generalized by the Two-Stage VRP with Profits and Buffers it is possible to use a consistent evaluation methodology.

The compared algorithms from the ISAVaN framework include algorithms that share similarities with 2BF-ILS and VNS_{OP} so that they can be expected to obtain promising results. The evaluation separately focused on the components $\texttt{Opt}_{\texttt{pre}}$

and $\texttt{Opt}_{\texttt{local}}$ from the framework that characterize an algorithm's search behavior. The results showed that there was not a single algorithm that performed best on all evaluation measures as the most suitable choice for the components $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$ depends on the considered time measure, the graphical evaluation measure and the optimization criterion (maximizing the total profit or minimizing the makespan).

In fact, for almost all considered values for the components $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}}$ specific instance properties and evaluation measures could be identified for which good results were obtained. For example, the local search procedure with subsequent testing of non-permutation solutions ($\texttt{Opt}_{\texttt{local}} = LS^*$) tended to perform well for the time measure $\texttt{tm} = SS$ (the number of tested subsets of nodes). Interestingly, algorithms with $\texttt{Opt}_{\texttt{pre}} = LK$ obtained fairly good results for the optimization problem of maximizing profit (similar to VNS_{OP} for the Orienteering Problem) and for the problem of minimizing makespan promising results were observed for $\texttt{Opt}_{\texttt{pre}} = NEH$, where the NEH heuristic is also commonly used for Two-Machine Flow Shops with Buffers.

In general, however, the most appropriate choice depends on the aforementioned factors, which influence whether the focus is placed on the number of evaluations or tested subsets, whether the speed of obtaining good solutions or the consistency of reaching a certain solution quality is more important, and whether the total profit or the makespan is the criterion to be optimized. For these questions, the proposed ISAVaN framework provides a spectrum of different algorithms forming a flexible and promising basis from which new methods for practical Vehicle Routing Problems can be developed.

Bibliography

[1] R. A. Abbaspour and F. Samadzadegan. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications* 38 (10), 12439–12452, 2011.

[2] S. Abdollahpour and J. Rezaeian. Minimizing Makespan for Flow Shop Scheduling Problem with Intermediate Buffers by Using Hybrid Approach of Artificial Immune System. *Applied Soft Computing* 28 (C), 44–56, 2015.

[3] A. Adewumi and O. Adeleke. A survey of recent advances in vehicle routing problems. *International Journal of System Assurance Engineering and Management* 9 (1), 155–172, 2018.

[4] A. Agnetis, F. Rossi, and G. Gristina. An exact algorithm for the batch sequencing problem in a two-machine flow shop with limited buffer. *Naval Research Logistics (NRL)* 45 (2), 141–164, 1998.

[5] T. J. Ai, J. Pribadi, and V. Ariyono. Solving the Team Orienteering Problem with Particle Swarm Optimization. *Industrial Engineering and Management Systems* 12, 198–206, 2013.

[6] S. Allen. A Two Stage Vehicle Routing Algorithm Applied to Disaster Relief Logistics after the 2015 Nepal Earthquake. *SIAM Undergraduate Research Online* 11, 2017.

[7] M. A. Aloulou, A. Bouzaiene, N. Dridi, and D. Vanderpooten. A bicriteria two-machine flow-shop serial-batching scheduling problem with bounded batch size. *Journal of Scheduling* 17 (1), 17–29, 2014.

[8] A. D. Amar and J. N. D. Gupta. Simulated Versus Real Life Data in Testing the Efficiency of Scheduling Algorithms. *IIE Transactions* 18 (1), 16–25, 1986.

[9] E. Angelelli, C. Archetti, C. Filippi, and M. Vindigni. The probabilistic orienteering problem. *Computers & Operations Research* 81, 269–281, 2017.

[10] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing* 15, 82–92, 2003.

[11] C. Archetti, N. Bianchessi, and M. Speranza. Optimal solutions for routing problems with profits. *Discrete Applied Mathematics* 161 (4), 547–557, 2013.

[12] C. Archetti, D. Feillet, A. Hertz, and M. G. Speranza. The undirected capacitated arc routing problem with profits. *Computers & Operations Research* 37 (11), 1860–1869, 2010.

[13] C. Archetti, M. G. Speranza, and D. Vigo. Vehicle Routing Problems with Profits. *Vehicle Routing*. Ed. by P. Toth and D. Vigo. Chap. 10, 273–297, 2014.

[14] H. Assimi, O. Harper, Y. Xie, A. Neumann, and F. Neumann. Evolutionary Bi-Objective Optimization for the Dynamic Chance-Constrained Knapsack Problem Based on Tail Bound Objectives. *ECAI 2020 - 24th European Conference on Artificial Intelligence, - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, 307–314, 2020.

[15] Y. Bao, L. Zheng, and H. Jiang. An Efficient Hybrid Based on HS and GA Solving Blocking Flow Shop Scheduling Problems. *Advanced Materials Research* 479, 1893–1896, 2012.

[16] J. Bautista, A. Cano, R. Companys, and I. Ribas. A bounded dynamic programming algorithm for the blocking flow shop problem. *2011 IEEE Workshop On Computational Intelligence In Production And Logistics Systems (CIPLS)*, 1–8, 2011.

[17] J. E. Beasley and E. M. Nascimento. The Vehicle Routing-Allocation Problem: A unifying framework. *Top* 4 (1), 65–86, 1996.

[18] J. Berlińska, K. Alexander, and Y. Zinder. Two-machine flow shop with dynamic storage space. *Optimization Letters* 15, 2433–2454, 2021.

[19] J. Berlińska, A. Kononov, and Y. Zinder. Two-Machine Flow Shop with a Dynamic Storage Space and UET Operations. *Optimization of Complex Systems: Theory, Models, Algorithms and Applications*. Ed. by H. A. Le Thi, H. M. Le, and T. Pham Dinh, 1139–1148, 2019.

[20] N. Bianchessi, R. Mansini, and M. G. Speranza. A branch-and-cut algorithm for the Team Orienteering Problem. *International Transactions in Operational Research* 25 (2), 627–635, 2018.

[21] Z. Borcinova. Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review* 8, 463–469, 2017.

[22] A. Bortfeldt and J. Homberger. Packing first, routing second – a heuristic for the vehicle routing and loading problem. *Computers & Operations Research* 40 (3), 873–885, 2013.

[23] H. Bouly, D.-C. Dang, and A. Moukrim. A Memetic Algorithm for the Team Orienteering Problem. *Applications of Evolutionary Computing*. Ed. by M. Giacobini, A. Brabazon, S. Cagnoni, G. A. Di Caro, R. Drechsler, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, J. McCormack, et al. Springer Berlin Heidelberg, Berlin, Heidelberg, 649–658, 2008.

[24] S. Boussier, D. Feillet, and M. Gendreau. An exact algorithm for team orienteering problems. *4OR quarterly journal of the Belgian, French and Italian Operations Research Societies* 5, 211–230, Sept. 2007.

[25] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[26] P. Brucker, S. Heitmann, and J. Hurink. Flow-shop problems with intermediate buffers. *Operations Research Spektrum* 25 (4), 549–574, 2003.

[27] T. Bulhões, M. Hà, R. Martinelli, and T. Vidal. The Vehicle Routing Problem with Service Level Constraints. *European Journal of Operational Research* 265 (2), 544–558, 2018.

[28] A. M. Campbell, M. Gendreau, and B. W. Thomas. The orienteering problem with stochastic travel and service times. *Annals of Operations Research* 186 (1), 61–81, 2011.

[29] V. Campos, R. Marti, J. Sánchez-Oro Calvo, and A. Duarte. GRASP with path relinking for the orienteering problem. *Journal of the Operational Research Society* 65, 1800–1813, 2014.

[30] C. Çetinkaya, I. Karaoglan, and H. Gökçen. Two-stage vehicle routing problem with arc time windows: A mixed integer programming formulation and a heuristic approach. *European Journal of Operational Research* 230 (3), 539–550, 2013.

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

[32] S. Chowdhury, M. Marufuzzaman, H. Tunc, L. Bian, and W. Bullington. A modified Ant Colony Optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance. *Journal of Computational Design and Engineering* 6 (3), 368–386, 2019.

[33] W. Cook, D. Applegate, R. Bixby, and V. Chvátal. *Concorde TSP Solver*. 2005. URL: http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm.

[34] B. D. Corwin. *Some Flow Shop Scheduling Problems Involving Sequence Dependent Setup Times*. PhD thesis. Cape Western Reserve University, 1969.

[35] B. D. Corwin and A. O. Esogbue. Two machine flow shop scheduling problems with sequence dependent setup times: A dynamic programming approach. *Naval Research Logistics Quarterly* 21 (3), 515–524, 1974.

[36] D.-C. Dang, R. Guibadj, and A. Moukrim. An effective PSO-inspired algorithm for the team orienteering problem. *European Journal of Operational Research* 229, 332–344, 2013.

[37] D.-C. Dang, R. El-Hajj, and A. Moukrim. A Branch-and-Cut Algorithm for Solving the Team Orienteering Problem. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Ed. by C. Gomes and M. Sellmann. Springer Berlin Heidelberg, Berlin, Heidelberg, 332–339, 2013.

[38] G. Deng, Z. Cui, and X. Gu. A discrete artificial bee colony algorithm for the blocking flow shop scheduling problem. *Proceedings of the 10th World Congress on Intelligent Control and Automation*, 518–522, 2012.

[39] J.-H. Duan, G.-Y. Qiao, and M. Zhang. Solving the Flow Shop Problem with Limited Buffers Using Differential Evolution. *Chinese Control and Decision Conference (CCDC)*, 2011.

[40] S. K. Dutta and A. A. Cunningham. Sequencing Two-Machine Flow-Shops with Finite Intermediate Storage. *Management Science* 21 (9), 989–996, 1975.

[41] E. Erkut and J. Zhang. The maximum collection problem with time-dependent rewards. *Naval Research Logistics* 43 (5), 749–763, 1996.

[42] A. Ernst, J. Fung, G. Singh, and Y. Zinder. Flexible flow shop with dedicated buffers. *Discrete Applied Mathematics* 261, 148–163, 2019.

[43] J. Faulin, A. Juan, F. Lera, and S. Grasman. Solving the Capacitated Vehicle Routing Problem with Environmental Criteria Based on Real Estimations in Road Transportation: A Case Study. *Procedia - Social and Behavioral Sciences* 20, 323–334, 2011.

[44] A. Federgruen and P. Zipkin. A Combined Vehicle Routing and Inventory Allocation Problem. *Operations Research* 32 (5), 1019–1037, 1984.

[45] D. Feillet, P. Dejax, and M. Gendreau. Traveling Salesman Problems With Profits: An Overview. *Transportation Science* 39, 188–205, 2001.

[46] V. Fernandez-Viagas and J. M. Framinan. NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research* 60, 27–36, 2015.

[47] J. Ferreira, A. Quintas, J. Oliveira, G. Pereira, and L. Dias. Solving the Team Orienteering Problem: Developing a Solution Tool Using a Genetic Algorithm Approach. *Advances in Intelligent Systems and Computing* 223, 365–375, 2014.

[48] F. V. Fomin and A. Lingas. Approximation algorithms for time-dependent orienteering. *Information Processing Letters* 83 (2), 57–62, 2002.

[49] Q. Fu, A. I. Sivakumar, and K. Li. Optimisation of flow-shop scheduling with batch processor and limited buffer. *International Journal of Production Research* 50 (8), 2267–2285, 2012.

[50] J. Fung and Y. Zinder. Permutation Schedules for a Two-machine Flow Shop with Storage. *Operation Research Letters* 44 (2), 153–157, 2016.

[51] J. Fung, Y. Zinder, and G. Singh. Flexible Flow Shop with Storage: Complexity and Optimisation Methods. *IFAC-PapersOnLine* 49, 237–242, 2016.

[52] N. G. Hall and C. Sriskandarajah. A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process. *Operations Research* 44, 510–525, 1996.

[53] D. Gale, H. W. Kuhn, and A. W. Tucker. Linear programming and the theory of games. *Activity analysis of production and allocation* 13, 317–335, 1951.

[54] M. R. Garey, D. S. Johnson, and R. Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research* 1 (2), 117–129, 1976.

[55] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[56] D. Gavalas, C. Konstantopoulos, and G. Pantziou. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics* 20, 291–328, 2014.

[57] P. Geser, H. T. Le, T. Hartmann, and M. Middendorf. On permutation schedules for two-machine flow shops with buffer constraints and constant processing times on one machine. *European Journal of Operational Research* 303 (2), 593–601, 2022.

[58] P. C. Gilmore and R. E. Gomory. Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Research* 12 (5), 655–679, 1964.

[59] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics* 34, 307–318, 1987.

[60] J. Grabowski and J. Pempera. The permutation flow shop problem with blocking. A tabu search approach. *Omega* 35, 302–311, 2007.

[61] R. Graham, E. Lawler, J. Lenstra, and A. R. Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics* 5, 287–326, 1979.

[62] P. Grangier, M. Gendreau, F. Lehuédé, and L.-M. Rousseau. The vehicle routing problem with cross-docking and resource constraints. *Journal of Heuristics* 27, 31–61, 2021.

[63] H. Gu, A. Kononov, J. Memar, and Y. Zinder. Efficient Lagrangian heuristics for the two-stage flow shop with job dependent buffer requirements. *Journal of Discrete Algorithms* 52-53, 143–155, 2018.

[64] A. Gunawan, H. C. Lau, and P. Vansteenwegen. Orienteering Problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research* 255 (2), 315–332, 2016.

[65] J. N. Gupta and W. P. Darrow. The two-machine sequence dependent flow-shop scheduling problem. *European Journal of Operational Research* 24 (3), 439–446, 1986.

[66] R. El-Hajj, R. Guibadj, A. Moukrim, and M. Serairi. A PSO based algorithm with an efficient optimal split procedure for the multiperiod vehicle routing problem with profit. *Annals of Operations Research* 291, 281–316, 2020.

[67] F. Hammami, M. Rekik, and L. C. Coelho. A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. *Computers & Operations Research* 123, 2020.

[68] P. Hansen, N. Mladenović, and J. M. Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175, 367–407, 2010.

[69] C. Hempsch and S. Irnich. Vehicle Routing Problems with Inter-Tour Resource Constraints. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Ed. by B. Golden, S. Raghavan, and E. Wasil. Springer US, Boston, MA, 421–444, 2008.

[70] Y.-C. Hsieh, P.-S. You, and C.-D. Liou. A note of using effective immune based approach for the flow shop scheduling with buffers. *Applied Mathematics and Computation* 215 (5), 1984–1989, 2009.

[71] T. İlhan, S. M. R. Iravani, and M. S. Daskin. The orienteering problem with stochastic profits. *IIE Transactions* 40 (4), 406–421, 2008.

[72] INRIX. *Durchschnittliche Geschwindigkeit* im Automobilverkehr in ausgewählten deutschen Städten im Jahr 2018 (in Meilen pro Stunde). In Statista*. 2019. URL: https://de.statista.com/statistik/daten/studie/994676/umfrage/innerstaedtische-durchschnittsgeschwindigkeit-im-autoverkehr-in-deutschen-staedten/.

[73] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly* 1 (1), 61–68, 1954.

[74] R. Kant and A. Mishra. The Orienteering Problem: A Review of Variants and Solution Approaches. *Proceedings of the 26th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2022)*, 41–46, 2022.

[75] I. Kara, P. S. Bicakci, and T. Derya. New Formulations for the Orienteering Problem. *Procedia Economics and Finance* 39, 849–854, 2016.

[76] M. Karlin and J. Heikkilä. *osm-graph-parser*. 2017. URL: https://github.com/rovaniemi/osm-graph-parser.

[77] S. Kataoka and S. Morito. An Algorithm for the Single Constraint Maximum Collection Problem. *Journal of the Operations Research Society in Japan*, 515–560, 1988.

[78] L. Ke, C. Archetti, and Z. Feng. Ants can solve the team orienteering problem. *Computers & Industrial Engineering* 54 (3), 648–665, 2008.

[79] M. Keshtkaran and K. Ziarati. An efficient evolutionary algorithm for the orienteering problem. *Journal of Heuristics* 22, 699–726, 2016.

[80] M. Keshtkaran, K. Ziarati, A. Bettinelli, and D. Vigo. Enhanced exact solution methods for the Team Orienteering Problem. *International Journal of Production Research* 54 (2), 591–601, 2016.

[81] M. R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E. Talbi. A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests. *Applied Soft Computing* 12 (4), 1426–1439, 2012.

[82] B.-I. Kim, H. Li, and A. L. Johnson. An augmented large neighborhood search method for solving the team orienteering problem. *Expert Systems with Applications* 40 (8), 3065–3072, 2013.

[83] H. Kise, T. Shioyama, and T. Ibaraki. Automated Two-machine Flowshop Scheduling: A Solvable Case. *IIE Transactions* 23 (1), 10–16, 1991.

[84] G. Kobeaga, M. Merino, and J. A. Lozano. An efficient evolutionary algorithm for the orienteering problem. *Computers & Operations Research* 90, 42–59, 2018.

[85] G. Kobeaga, M. Merino, and J. A. Lozano. *OPLib: Test instances for the Orienteering Problem*. 2018. URL: https://github.com/bcamath-ds/OPLib/tree/master/instances.

[86] A. Kononov, J.-S. Hong, P. Kononova, and F.-C. Lin. Quantity-based buffer-constrained two-machine flowshop problem: active and passive prefetch models for multimedia applications. *Journal of Scheduling* 15 (4), 487–497, 2012.

[87] A. Kononov, J. Memar, and Y. Zinder. Flow Shop with Job–Dependent Buffer Requirements – a Polynomial–Time Algorithm and Efficient Heuristics. *MOTOR 2019: Mathematical Optimization Theory and Operations Research*, 342–357, 2019.

[88] P. A. Kononova and Y. A. Kochetov. The Variable Neighborhood Search for the Two Machine Flow Shop Problem with a Passive Prefetch. *Journal of Applied and Industrial Mathematics* 7 (1), 54–67, 2013.

[89] H. Kuhn, D. Schubert, and A. Holzapfel. Integrated order batching and vehicle routing operations in grocery retail – A General Adaptive Large Neighborhood Search algorithm. *European Journal of Operational Research* 294 (3), 1003–1021, 2021.

[90] E. Kupfer, H. T. Le, J. Zitt, Y.-C. Lin, and M. Middendorf. A Hierarchical Simple Probabilistic Population-Based Algorithm Applied to the Dynamic TSP. *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–8, 2021.

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

[92] T. Le Hoang. A Fuzzy Local Grid Refinement Method for Sparse-Grid-Based Function Approximations. *Proceedings of 6th International Conference of Young Scientists on Solutions of Applied Problems in Control and Communications, Data Processing and Data Analysis*. Ed. by U. Fissgus, B. Krause, A. Kostygov, A. Petrochenkov, and L. Mylnikov, 116–121, 2015.

[93] H. T. Le, P. Geser, and M. Middendorf. An Iterated Local Search Algorithm for the Two-Machine Flow Shop Problem with Buffers and Constant Processing Times on One Machine. *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2019)*. Ed. by A. Liefooghe and L. Paquete, 50–65, 2019.

[94] H. T. Le. *2SVRP_Results*. GitHub repository. 2022. URL: https://github.com/L-HT/2SVRP_Results.

[95] H. T. Le. *Dynamic Orienteering Algorithms*. GitHub repository. 2020. URL: https://github.com/L-HT/DynamicOrienteeringAlgorithms/.

[96] H. T. Le, P. Geser, and M. Middendorf. Iterated Local Search and Other Algorithms for Buffered Two-Machine Permutation Flow Shops with Constant Processing Times on One Machine. *Evolutionary Computation* 29 (3), 415–439, 2021.

[97] H. T. Le, M. Middendorf, and Y. Shi. An Improvement Heuristic Based on Variable Neighborhood Search for a Dynamic Orienteering Problem. *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2021)*. Ed. by C. Zarges and S. Verel, 68–83, 2021.

[98] H. T. Le, M. Middendorf, and Y. Shi. An Improvement Heuristic Based on Variable Neighborhood Search for Dynamic Orienteering Problems with Changing Node Values and Changing Budgets. *SN Computer Science* 3 (4), *Bio-inspired Algorithms for Combinatorial Optimization*, 326, 2022.

[99] D. Lee and J. Ahn. Vehicle routing problem with vector profits with max-min criterion. *Engineering Optimization* 51 (2), 352–367, 2019.

[100] R. Leisten. Flowshop sequencing problems with limited buffer storage. *International Journal of Production Research* 28 (11), 2085–2100, 1990.

[101] J. Li and W. Lu. Full truckload vehicle routing problem with profits. *Journal of Traffic and Transportation Engineering (English Edition)* 1 (2), 146–152, 2014.

[102] J. Li and Q.-K. Pan. Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences* 316, 487–502, 2015.

[103] K. Li and H. Tian. A two-level self-adaptive variable neighborhood search algorithm for the prize-collecting vehicle routing problem. *Applied Soft Computing* 43, 469–479, 2016.

[104] S. Li and L. Tang. A tabu search algorithm based on new block properties and speed-up method for permutation flow-shop with finite intermediate storage. *Journal of Intelligent Manufacturing* 16 (4), 463–477, 2005.

[105] J. J. Liang, Q.-K. Pan, and T.-J. Chen. A Dynamic Multi-swarm Particle Swarm Optimizer for blocking flow shop scheduling. *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, 323–327, 2010.

[106] C. J. Liao, L. M. Liao, and C. T. Tseng. A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research* 44 (20), 4297–4309, 2006.

[107] F.-C. Lin, J.-S. Hong, and B. M. T. Lin. A Two-machine Flowshop Problem with Processing Time-dependent Buffer constraints–An Application in Multimedia Presentations. *Computers & Operations Research* 36 (4), 1158–1175, 2009.

[108] F.-C. Lin, J.-S. Hong, and B. M. Lin. Sequence optimization for media objects with due date constraints in multimedia presentations from digital libraries. *Information Systems* 38 (1), 2013.

[109] B. Liu, L. Wang, and Y.-H. Jin. An Effective Hybrid PSO-based Algorithm for Flow Shop Scheduling with Limited Buffers. *Computers & Operations Research* 35 (9), 2791–2806, 2008.

[110] S. Q. Liu and E. Kozan. Scheduling a flow-shop with combined buffer conditions. *International Journal of Production Economics* 117 (2), 371–380, 2009.

[111] W. Liu, Y. Jin, and M. Price. A new improved NEH heuristic for permutation flowshop scheduling problems. *International Journal of Production Economics* 193, 21–30, 2017.

[112] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3, 43–58, 2016.

[113] M. Mann, B. Zion, D. Rubinstein, R. Linker, and I. Shmulevich. The Orienteering Problem with Time Windows Applied to Robotic Melon Harvesting. *Journal of Optimization Theory and Applications* 168, 1–22, 2015.

[114] Y. Marinakis, M. Politis, M. Marinaki, and N. Matsatsinis. A Memetic-GRASP Algorithm for the Solution of the Orienteering Problem. *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Ed. by H. A. Le Thi, T. Pham Dinh, and N. T. Nguyen, 105–116, 2015.

[115] S. Martinez, S. Dauzère-Pérès, C. Guéret, Y. Mati, and N. Sauer. Complexity of Flowshop Scheduling Problems with a New Blocking Constraint. *European Journal of Operational Research* 169, 855–864, 2006.

[116] L. d. C. Martins, R. D. Tordecilla, J. Castaneda, A. A. Juan, and J. Faulin. Electric Vehicle Routing, Arc Routing, and Team Orienteering Problems in Sustainable Transportation. *Energies* 14 (16), 2021.

[117] Y. Min, B. C. Choi, and M. J. Park. Two-machine flow shops with an optimal permutation schedule under a storage constraint. *Journal of Scheduling* 23, 327–336, 2019.

[118] M. Mısır, A. Gunawan, and P. Vansteenwegen. Algorithm Selection for the Team Orienteering Problem. *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2022)*. Ed. by L. Pérez Cáceres and S. Verel, 33–45, 2022.

[119] G. Moslehi and D. Khorasanian. A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion. *Computers & Operations Research* 52, 260–268, 2014.

[120] V. Nagarajan and R. Ravi. The Directed Orienteering Problem. *Algorithmica* 60, 1017–1030, 2011.

[121] S. Nanda Kumar and R. Panneerselvam. A Survey on the Vehicle Routing Problem and Its Variants. *Intelligent Information Management* 4 (3), 66–74, 2012.

[122] M. Nawaz, E. Emory Enscore, and I. Ham. A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem. *Omega* 11, 91–95, 1983.

[123] T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6, 1–24. ISSN: 2210-6502, 2012.

[124] E. Nowicki. The permutation flow shop with buffers: A tabu search approach. *European Journal of Operational Research* 116 (1), 205–219, 1999.

[125] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. 2020. URL: https://www.openstreetmap.org.

[126] M. Ostermeier, A. Holzapfel, H. Kuhn, and D. Schubert. Integrated zone picking and vehicle routing operations with restricted intermediate storage. *OR Spectrum* 44, 795–832, 2022.

[127] K. Ostrowski, J. Karbowska-Chilinska, J. Koszelew, and P. Zabielski. Evolution-inspired local improvement algorithm solving orienteering problem. *Annals of Operations Research* 253, 519–543, 2017.

[128] Q.-K. Pan, L. Wang, H.-Y. Sang, J.-Q. Li, and M. Liu. A High Performing Memetic Algorithm for the Flowshop Scheduling Problem With Blocking. *IEEE Transactions on Automation Science and Engineering* 10 (3), 741–756, 2013.

[129] Q.-K. Pan, L. Wang, and L. Gao. A chaotic harmony search algorithm for the flow shop scheduling problem with limited buffers. *Applied Soft Computing* 11, 5270–5280, 2011.

[130] Q.-K. Pan, L. Wang, L. Gao, and W. D. Li. An Effective Hybrid Discrete Differential Evolution Algorithm for the Flow Shop Scheduling with Intermediate Buffers. *Information Science* 181 (3), 668–685, 2011.

[131] C. H. Papadimitriou and P. C. Kanellakis. Flowshop Scheduling with Limited Temporary Storage. *Journal of the Association for Computing Machinery (J ACM)* 27 (3), 533–549, 1980.

[132] V. Papapanagiotou, R. Montemanni, and L. M. Gambardella. Comparison of Objective Function Evaluators for a Stochastic Orienteering Problem. *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, 465–471, 2016.

[133] G. Perboli, R. Tadei, and D. Vigo. The Two-Echelon Capacitated Vehicle Routing Problem: Models and Math-Based Heuristics. *Transportation Science* 45 (3), 364–380, 2011.

[134] A. Pirabán-Ramírez, W. J. Guerrero-Rueda, and N. Labadie. The multi-trip vehicle routing problem with increasing profits for the blood transportation: An iterated local search metaheuristic. *Computers & Industrial Engineering* 170, 108294, 2022.

[135] M. Pranzo. Batch scheduling in a two-machine flow shop with limited buffer and sequence independent setup times and removal times. *European Journal of Operational Research* 153 (3), 581–592, 2004.

[136] B. Qian, L. Wang, D. X. Huang, and X. Wang. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. *International Journal of Production Research* 47 (1), 1–24, 2009.

[137] B. Qian, L. Wang, D.-X. Huang, W.-l. Wang, and X. Wang. An Effective Hybrid DE-based Algorithm for Multi-objective Flow Shop Scheduling with Limited Buffers. *Computers & Operations Research* 36 (1), 209–233, 2009.

[138] G. Rabadi, M. K. Msakni, E. Rodriguez-Velasquez, and W. Alvarez-Bermudez. New characteristics of optimal solutions for the two-machine flowshop problem with unlimited buffers. *Journal of the Operational Research Society* 70 (6), 962–973, 2019.

[139] S. S. Reddi and C. V. Ramamoorthy. On the Flow-Shop Sequencing Problem with No Wait in Process. *Journal of the Operational Research Society* 23 (3), 323–331, 1972.

[140] C. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research* 22 (1), 5–13, 1995.

[141] I. Ribas, R. Companys, and X. Tort-Martorell. Efficient Heuristics for the Parallel Blocking Flow Shop Scheduling Problem. *Expert Systems with Applications* 74 (C), 41–54, 2017.

[142] D. P. Ronconi. A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics* 87, 2004.

[143] V. Roostapour, A. Neumann, and F. Neumann. On the Performance of Baseline Evolutionary Algorithms on the Dynamic Knapsack Problem. *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, Part I*, 158–169, 2018.

[144] V. Roostapour, A. Neumann, and F. Neumann. Single- and multi-objective evolutionary algorithms for the knapsack problem with dynamically changing constraints. *Theoretical Computer Science* 924, 129–147, 2022.

[145] D. A. Rossit, Ó. C. Vásquez, F. Tohmé, M. Frutos, and M. D. Safe. A combinatorial analysis of the permutation and non-permutation flow shop scheduling problems. *European Journal of Operational Research* 289 (3), 841–854, 2021.

[146] D. A. Rossit, F. Tohmé, and M. Frutos. The Non-Permutation Flow-Shop scheduling problem: A literature review. *Omega* 77, 143–153, 2018.

[147] D. A. Rossit, F. Tohmé, M. Frutos, M. Safe, and Ó. Vásquez. Critical paths of non-permutation and permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations* 11, 281–298, 2 2020.

[148] Rotenberg (https://math.stackexchange.com/users/242055/rotenberg). *Reduction from Hamiltonian cycle to Hamiltonian path*. Mathematics Stack Exchange. 2016. URL: https://math.stackexchange.com/q/1290804.

[149] H.-Y. Sang and Q.-K. Pan. An effective invasive weed optimization algorithm for the flow shop scheduling with intermediate buffers. *25th Chinese Control and Decision Conference (CCDC)*, 861–864, 2013.

[150] B. Sarasola, M. R. Khouadjia, E. Alba, L. Jourdan, and E. Talbi. Flexible Variable Neighborhood Search in Dynamic Vehicle Routing. *Applications of Evolutionary Computation (EvoApplications 2011)*, 344–353, 2011.

[151] M. Schilde, K. F. Doerner, R. F. Hartl, and G. Kiechle. Metaheuristics for the bi-objective orienteering problem. *Swarm Intelligence* 3, 179–201, 2009.

[152] J. P. Schmitt, F. Baldo, and R. S. Parpinelli. A MAX-MIN Ant System with Short-Term Memory Applied to the Dynamic and Asymmetric Traveling Salesman Problem. *7th Brazilian Conference on Intelligent Systems (BRACIS 2018)*, 1–6, 2018.

[153] W. Schneider. *BBBike.org*. 2020. URL: https://download.bbbike.org/osm/.

[154] D. Schubert. *Integrated Order Picking and Vehicle Routing Operations – Literature Review and Further Research Opportunities*. 2020. URL: https://ssrn.com/abstract=3631748.

[155] Z. Shen, M. Dessouky, and F. Ordóñez. A two-stage vehicle routing model for large-scale bioterrorism emergencies. *Networks* 54 (4), 255–269, 2009.

[156] J. Simons. Heuristics in flow shop scheduling with sequence dependent setup times. *Omega* 20 (2), 215–225, 1992.

[157] C. Smutnicki. A two-machine permutation flow shop scheduling problem with buffers. *Operations Research Spektrum* 20, 229–235, 1998.

[158] S. A. Soltani and B. Karimi. Cyclic hybrid flow shop scheduling problem with limited buffers and machine eligibility constraints. *The International Journal of Advanced Manufacturing Technology* 76 (9), 1739–1755, 2015.

[159] W. Souffriau, P. Vansteenwegen, and G. Vanden Berghe. A Path Relinking approach for the Team Orienteering Problem. *Computers & Operations Research* 37, 1853–1859, 2010.

[160] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden. A greedy randomised adaptive search procedure for the team orienteering problem. *EU/MEeting 2008 on metaheuristics for logistics and vehicle routing*, 1–9, 2008.

[161] F. Stavropoulou, P. Repoussis, and C. Tarantilis. The Vehicle Routing Problem with Profits and consistency constraints. *European Journal of Operational Research* 274 (1), 340–356, 2019.

[162] A. Stenger. The prize-collecting vehicle routing problem with non-linear cost: Integration of Subcontractors into Route Design of Small Package Shippers. *Proceedings of the 1st International Conference on Operations Research and Enterprise Systems (ICORES 2012)*, 265–273, 2012.

[163] A. Stenger, M. Schneider, and D. Goeke. The prize-collecting vehicle routing problem with single and multiple depots and non-linear cost. *EURO Journal on Transportation and Logistics* 2, 57–87, 2013.

[164] L. Strak, R. Skinderowicz, U. Boryczka, and A. Nowakowski. A Self-Adaptive Discrete PSO Algorithm with Heterogeneous Parameter Values for Dynamic TSP. *Entropy* 21 (8), 738, 2019.

[165] V. Strusevich and P. Zwaneveld. On non-permutation solutions to some two machine flow shop scheduling problems. *Mathematical Methods of Operations Research* 39 (3), 305–319, 1994.

[166] Y. Sun, S. Wang, Y. Shen, X. Li, A. T. Ernst, and M. Kirley. Boosting Ant Colony Optimization via Solution Prediction and Machine Learning. *arXiv*. URL: https://arxiv.org/abs/2008.04213, 2020.

[167] W. Szeto, Y. Wu, and S. C. Ho. An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research* 215 (1), 126–135, 2011.

[168] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64, 278–285, 1993.

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

[170] L. Tang and X. Wang. Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem. *The International Journal of Advanced Manufacturing Technology* 29, 1246–1258, 2006.

[171] Z. Tao, X. Liu, and P. Zeng. Study on Hybrid Flow Shop Scheduling Problem with Blocking Based on GASA. *The Open Automation and Control Systems Journal* 6, 593–600, 2014.

[172] M. F. Tasgetiren, D. Kizilay, Q.-K. Pan, and P. Suganthan. Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Computers & Operations Research* 77, 111–126, 2017.

[173] L. H. Thanh. A Learning Algorithm Based on λ-Policy Iteration and Its Application to the Video Game "Tetris Attack". *46. Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2016), Klagenfurt, Austria*. Ed. by H. C. Mayr and M. Pinzger. Gesellschaft für Informatik e.V., Bonn, 2157–2162, 2016.

[174] T. C. Thayer and S. Carpin. Solving Large-scale Stochastic Orienteering Problems with Aggregation. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020)*, 2452–2458, 2020.

[175] A. Thibbotuwawa, G. Bocewicz, P. Nielsen, and Z. Banaszak. Unmanned Aerial Vehicle Routing Problems: A Literature Review. *Applied Sciences* 10 (13), 2020.

[176] D. Trachanatzi, M. Rigakis, M. Marinaki, and Y. Marinakis. A firefly algorithm for the environmental prize-collecting vehicle routing problem. *Swarm and Evolutionary Computation* 57, 100712, 2020.

[177] T. H. Tran and K. M. Ng. A water-flow algorithm for flexible flow shop scheduling with intermediate buffers. *Journal of Scheduling* 14 (5), 483–500, 2011.

[178] E. Tsakirakis, M. Marinaki, Y. Marinakis, and N. Matsatsinis. A similarity hybrid harmony search algorithm for the Team Orienteering Problem. *Applied Soft Computing* 80, 776–796, 2019.

[179] T. Tsiligiridis. Heuristic Methods Applied to Orienteering. *Journal of the Operational Research Society* 35, 797–809, 1984.

[180] E. Vallada, R. Ruiz, and J. M. Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research* 240, 666–677, 2015.

[181] J. Van Belle, P. Valckenaers, and D. Cattrysse. Cross-docking: State of the art. *Omega* 40 (6), 827–846, 2012.

[182] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden. The City Trip Planner: An expert system for tourists. *Expert Systems with Applications* 38 (6), 6540–6546, 2011.

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

[184] J. A. Vázquez-Rodríguez and G. Ochoa. On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society* 62 (2), 381–396, 2011.

[185] T. Vidal, G. Laporte, and P. Matl. A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research* 286 (2), 401–416, 2020.

[186] T. Vidal, N. Maculan, L. S. Ochi, and P. H. Vaz Penna. Large Neighborhoods with Implicit Customer Selection for Vehicle Routing Problems with Profits. *Transportation Science* 50 (2), 720–734, 2016.

[187] A. Viktorin, D. Hrabec, and M. Pluhacek. Multi-Chaotic Differential Evolution For Vehicle Routing Problem With Profits. *30th European Conference on Modelling and Simulation*, 245–251, 2016.

[188] H. Wang, W. Wang, H. Sun, C. Li, S. Rahnamayan, and Y. Liu. A modified cuckoo search algorithm for flow shop scheduling problem with blocking. *2015 IEEE Congress on Evolutionary Computation (CEC)*, 456–463, 2015.

[189] L. Wang, Q.-K. Pan, P. N. Suganthan, W.-H. Wang, and Y.-M. Wang. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research* 37 (3), 509–520, 2010.

[190] L. Wang, Q.-K. Pan, and M. F. Tasgetiren. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. *Computers & Industrial Engineering* 61 (1), 76–83, 2011.

[191] L. Wang, L. Zhang, and D.-Z. Zheng. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research* 33, 2960–2971, 2006.

[192] X. Wang, B. L. Golden, and E. A. Wasil. Using a Genetic Algorithm to Solve the Generalized Orienteering Problem. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Ed. by B. Golden, S. Raghavan, and E. Wasil. Springer US, Boston, MA, 263–274, 2008.

[193] J.-P. Watson, L. Barbulescu, L. D. Whitley, and A. E. Howe. Contrasting Structured and Random Permutation Flow-Shop Scheduling Problems: Search-Space Topology and Algorithm Performance. *INFORMS Journal on Computing* 14 (2), 98–123, 2002.

[194] H. Wei and J. Yuan. Two-machine flow-shop scheduling with equal processing time on the second machine for minimizing total weighted completion time. *Operations Research Letters* 47 (1), 41–46, 2019.

[195] T. Weise, R. Chiong, J. Lassig, K. Tang, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao. Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem. *IEEE Computational Intelligence Magazine* 9 (3), 40–52, 2014.

[196] W. Xu, Z. Xu, J. Peng, W. Liang, T. Liu, X. Jia, and S. K. Das. Approximation Algorithms for the Team Orienteering Problem. *IEEE Conference on Computer Communications (IEEE INFOCOM 2020)*, 1389–1398, 2020.

[197] Your Europe. *Road transportation workers*. 2020. URL: https://europa.eu/youreurope/business/human-resources/transport-sector-workers/road-transportation-workers.

[198] Y. Yu and T. Li. Scheduling a constrained hybrid flow shop problem by heuristic algorithm. *26th Chinese Control and Decision Conference (CCDC)*, 2532–2537, 2014.

[199] G. Zhang and K. Xing. Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion. *Computers and Operations Research* 108, 33–43, 2019.

[200] S.-J. Zhang and X.-S. Gu. An effective discrete artificial bee colony algorithm for flow shop scheduling problem with intermediate buffers. *Journal of Central South University* 22 (9), 3471–3484, 2015.

[201] S. Zhang, J. W. Ohlmann, and B. W. Thomas. Dynamic Orienteering on a Network of Queues. *Transportation Science* 52 (3), 691–706, 2018.

[202] T. Zhang, W. Chaovalitwongse, Y.-J. Zhang, and P. Pardalos. The hot-rolling batch scheduling method based on the prize collecting vehicle routing problem. *Journal of Industrial and Management Optimization* 5, 749–765, 2009.

[203] Z. Zhang, H. Qin, and Y. Li. Multi-Objective Optimization for the Vehicle Routing Problem With Outsourcing and Profit Balancing. *IEEE Transactions on Intelligent Transportation Systems* 21 (5), 1987–2001, 2020.

[204] H. Zhong, R. W. Hall, and M. Dessouky. Territory Planning and Vehicle Dispatching with Driver Learning. *Transportation Science* 41 (1), 74–89, 2007.

[205] Y. Zinder, A. Kononov, and J. Fung. A 5-parameter complexity classification of the two-stage flow shop scheduling problem with job dependent storage requirements. *Journal of Combinatorial Optimization*, 1–34, 2021.

[206] X. Zou, L. Liu, K. Li, and W. Li. A coordinated algorithm for integrated production scheduling and vehicle routing problem. *International Journal of Production Research* 56 (15), 5005–5024, 2018.

List of Publications

- H. T. Le, M. Middendorf, and Y. Shi. An Improvement Heuristic Based on Variable Neighborhood Search for Dynamic Orienteering Problems with Changing Node Values and Changing Budgets. *SN Computer Science* 3 (4), *Bio-inspired Algorithms for Combinatorial Optimization*, 326, 2022.
- P. Geser, H. T. Le, T. Hartmann, and M. Middendorf. On permutation schedules for two-machine flow shops with buffer constraints and constant processing times on one machine. *European Journal of Operational Research* 303 (2), 593–601, 2022.
- E. Kupfer, H. T. Le, J. Zitt, Y.-C. Lin, and M. Middendorf. A Hierarchical Simple Probabilistic Population-Based Algorithm Applied to the Dynamic TSP. *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–8, 2021.
- H. T. Le, M. Middendorf, and Y. Shi. An Improvement Heuristic Based on Variable Neighborhood Search for a Dynamic Orienteering Problem. *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2021)*. Ed. by C. Zarges and S. Verel, 68–83, 2021.
- H. T. Le, P. Geser, and M. Middendorf. Iterated Local Search and Other Algorithms for Buffered Two-Machine Permutation Flow Shops with Constant Processing Times on One Machine. *Evolutionary Computation* 29 (3), 415–439, 2021.
- H. T. Le, P. Geser, and M. Middendorf. An Iterated Local Search Algorithm for the Two-Machine Flow Shop Problem with Buffers and Constant Processing Times on One Machine. *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2019)*. Ed. by A. Liefooghe and L. Paquete, 50–65, 2019.
- L. H. Thanh. A Learning Algorithm Based on λ-Policy Iteration and Its Application to the Video Game "Tetris Attack". *46. Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2016), Klagenfurt, Austria*. Ed. by H. C. Mayr and M. Pinzger. Gesellschaft für Informatik e.V., Bonn, 2157–2162, 2016.
- T. Le Hoang. A Fuzzy Local Grid Refinement Method for Sparse-Grid-Based Function Approximations. *Proceedings of 6th International Conference of Young Scientists on Solutions of Applied Problems in Control and Communications, Data Processing and Data Analysis*. Ed. by U. Fissgus, B. Krause, A. Kostygov, A. Petrochenkov, and L. Mylnikov, 116–121, 2015.

List of Figures

2.1 Visualization of an example graph with jobs for the Two-Stage VRP with Profits and Buffers . . . 18
2.2 Visualization of an example route for an instance of the Two-Stage VRP with Profits and Buffers . . . 22

3.1 Example instance for the Two-Stage VRP with Profits and Buffers with the additional property that travel times on all edges are zero . 37
3.2 Relations between algorithms . . . 38
3.3 Visualization for Theorem 3.2.1 . . . 46
3.4 Structure of an optimal schedule for the case $F2|\mathsf{schedType}, b_J = c, \mathit{intermediateBuffer}, s_J = a_J, c \geq a_J|C_{\max}$ . . . 56
3.5 Structure of an optimal schedule for the case $F2|\mathsf{schedType}, b_J = c, \mathit{intermediateBuffer}, s_J = a_J, c \leq a_J|C_{\max}$ . . . 57
3.6 Structure of an optimal (non-permutation) schedule σ^* for an instance $\mathcal{I}_k$ (spanning buffer) . . . 64
3.7 Structure of a best permutation schedule σ^* for an instance $\mathcal{I}_k$ (spanning buffer) . . . 64
3.8 Diagram to visualize the different "states" when scheduling a job j as the next job in a feasible permutation schedule σ^P and the incurred idle times on M_1 and M_2 for the instances $\mathcal{I}_k$ . . . 67
3.9 Structure of a best permutation schedule σ^*_{perm} for the instances $\mathcal{I}_k$ (intermediate buffer) . . . 69
3.10 Structure of an optimal non-permutation schedule σ^* for the instances $\mathcal{I}_k$ (intermediate buffer) . . . 69

3.11 Progress curve for an instance using the spanning buffer, $s_J = 1$, uniformly random processing times and 150 jobs (left) and an instance with spanning buffer, $s_J = a_J$ for all J, normally distributed processing times and 100 jobs (right) 83
3.12 Scatter plot of RPD values of the final solution calculated by 2BF-ILS in relation to the respective RPD values of the other algorithms . . . 85
3.13 Distribution of run times between the standard NEH heuristic and 2BF-OPT on instances of the type $F2|prmu, b_i = c, spanningBuffer, s_i = a_i|C_{\max}$ for instances with $c \leq a_J$ (left) and $c \geq a_J$ (right) for all J . . . 86

4.1 Example instance for the Two-Stage VRP with Profits and Bufferswith the additional property that processing times for all jobs are zero . . 90
4.2 Examples for progress curves and an example for an Orienteering Problem instance 108
4.3 Visualization of ECDFs for the instance `LGF-Berlin-150-80000` and `LGF-Berlin-100-80000-1` with respect to the time measure $\texttt{tm} = SS$ 115
4.4 Scatter plot of average RPD values for the solution quality of VNS_{OP} at $t = 100,000\,FE$ (left) and at the end of the run (right) in relation to the respective RPD values of the other algorithms. 117

5.1 Example instance for Lemma 5.1.9 143
5.2 Example diagrams for progress curves and empirical cumulative distribution functions (ECDF) 176
5.3 Left: Visualization of progress curves for the $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instance `LGF-gr120-gen4-85.oplib-590` (with $n = 120$ and a spanning buffer) and the algorithms with $\texttt{Opt}_{\texttt{local}} = noLS$. Right: Scatter plot of average RPD values for the solution quality with $\texttt{Opt}_{\texttt{pre}} = LK$ at the end of the run in relation to the respective RPD values of algorithms with different $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}} = noLS$ 178
5.4 Left: Left: Scatter plot of average RPD values for the solution quality with $\texttt{Opt}_{\texttt{local}} = LS^*$ at the end of the run in relation to the respective RPD of algorithms with different values for $\texttt{Opt}_{\texttt{local}}$ and $\texttt{Opt}_{\texttt{pre}} = none$.Right: Results of the pairwise comparisons between different $\texttt{Opt}_{\texttt{local}}$ using the two-sided sign test over all $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instances. 181

5.5 Left: Visualization of progress curves for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Leipzig-150-80000-2` and the algorithms with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$. Right: ECDF curves for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Berlin-50-80000-1` and for the algorithms with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$. 183

5.6 Left: Scatter plot of average RPD values for the solution quality with $\texttt{Opt}_{\texttt{pre}} = \textit{NEH}$ at the end of the run in relation to the respective RPD values of algorithms with different $\texttt{Opt}_{\texttt{pre}}$ and $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$. Right: ECDF curves for the $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instance `LGF-Berlin-100-80000-1` for the algorithms with $\texttt{Opt}_{\texttt{pre}} = \textit{none}$. 185

List of Tables

3.1 Overview of metaheuristics that have been applied to flow shops with limited buffers ("Limited Buffer FS") as well as blocking flow shops ("Blocking FS"). 41
3.2 Results for the configuration of numerical parameters for 2BF-ILS, HVNS and DABC as calculated by `irace` 79
3.3 Relative area under curve (AUC) values for PC diagrams which were calculated for each algorithm as well as each time measure and averaged over different subsets of instances 81
3.4 Relative area under curve (AUC) values for ECDF curves, averaged over different sets of instances 82
3.5 Results of the pairwise comparisons between the algorithms using the two-sided sign test for both sets of flow shop problems 84

4.1 Overview of the evaluation metrics used during the computational experiments. 112
4.2 Average values for $AUC_{I,A}^{norm}$ for the progress curve diagrams per time measure, aggregated over different subsets of instances 113
4.3 Average values for $AUC_{I,A}^{norm}$ with respect to ECDF diagrams per time measure, aggregated over different subsets of instances 113
4.4 Results of the pairwise comparisons between the algorithms using the two-sided sign test over all Orienteering Problem instances ($n = 22$) at certain points in time. 116

5.1 Overview of the theoretical properties for the Two-Stage VRP with Profits and Buffers and its subcases regarding computational complexity and potential gaps between permutation schedules and non-permutation schedules 160

5.2 Average $AUC_{I,A}^{norm}$ for different values of $\texttt{Opt}_{\texttt{pre}}$ with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ on $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances 177
5.3 Average $AUC_{I,A}^{norm}$ for different values of $\texttt{Opt}_{\texttt{local}}$ with $\texttt{Opt}_{\texttt{pre}} = \textit{none}$ on $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances 179
5.4 Average $AUC_{I,A}^{norm}$ for different values of $\texttt{Opt}_{\texttt{pre}}$ with $\texttt{Opt}_{\texttt{local}} = \textit{noLS}$ on $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances 182
5.5 Average $AUC_{I,A}^{norm}$ for different values of $\texttt{Opt}_{\texttt{local}}$ with $\texttt{Opt}_{\texttt{pre}} = \textit{none}$ on $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ instances with respect to PC diagrams (upper half) and ECDF diagrams (lower half) per time measure, aggregated over different subsets of instances. 184

List of Algorithms

3.1 2BF-OPT for $F2|prmu, b_J = c, spanningBuffer, s_J = a_J, c \geq a_J|C_{\max}$ . . . 53
3.2 Modified NEH heuristic (mNEH) . . . 73
3.3 2BF-ILS . . . 75

4.1 VNS_{OP} . . . 105

5.1 Local Search subroutine of 2BF-ILS . . . 169
5.2 General structure of an algorithm in the ISAVaN framework . . . 170
5.3 Initialization routine `Init` used for $\mathcal{O}_{\min C_{\max},\, R \geq Q}$ . . . 171
5.4 Initialization routine `Init` used for $\mathcal{O}_{\max R,\, C_{\max} \leq B}$ . . . 171

Bibliographische Beschreibung:

Le, Hoang Thanh

Two-Stage Vehicle Routing Problems with Profits and Buffers: Analysis and Metaheuristic Optimization Algorithms

Universität Leipzig, Dissertation

220 S., 206 Lit., 25 Abb., 14 Tab.

Wissenschaftlicher Werdegang

Juli 2012	Ludwigsgymnasium Köthen Erwerb der allgemeinen Hochschulreife
Oktober 2012–März 2016	Hochschule Anhalt Studiengang: Angewandte Informatik Abschluss: Bachelor of Science
April 2016–März 2018	Universität Leipzig Studiengang: Informatik Abschluss: Master of Science
seit April 2018	Universität Leipzig Wissenschaftlicher Mitarbeiter
Forschungsgruppe:	Schwarmintelligenz und Komplexe Systeme Fakultät für Mathematik und Informatik

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

Ort: _____

Datum: Hoang Thanh Le