FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# A Variable Neighborhood Search for the Job Sequencing with One Common and Multiple Secondary Resources Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Software Engineering & Internet Computing**

eingereicht von

**Thomas Kaufmann, BSc.**
Matrikelnummer 01129115

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Dipl.-Ing. Matthias Horn, BSc.

Wien, 2. Dezember 2019

| Thomas Kaufmann | Günther Raidl |
|---|---|

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# A Variable Neighborhood Search for the Job Sequencing with One Common and Multiple Secondary Resources Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering & Internet Computing**

by

**Thomas Kaufmann, BSc.**
Registration Number 01129115

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Dipl.-Ing. Matthias Horn, BSc.

Vienna, 2nd December, 2019

Thomas Kaufmann | Günther Raidl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

# Erklärung zur Verfassung der Arbeit

Thomas Kaufmann, BSc.
Gredlerstraße 9/10, 1020 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. Dezember 2019

Thomas Kaufmann

# Acknowledgements

I want to thank my family and friends for their continuous support throughout my life and particularly during my studies. Furthermore, I want to express my gratitude to my supervisors Prof. Raidl and especially Matthias Horn for their guidance, support and immediate feedback throughout this work.

# Abstract

In this thesis we are interested in heuristically solving a problem appearing in novel cancer treatment facilities, where a particle beam serves multiple treatment rooms in a multiplexed manner. Previously, several variations of the induced optimization problem have been studied intensively, however, in this work, we solely concentrate on the subproblem of finding feasible daily schedules, as recently introduced by Horn et al. In particular, we are interested in scheduling a set of jobs without preemption while minimizing the objective function. Each job needs two types of resources. The *common* resource is shared among all jobs but is acquired only for a certain period in a job's processing time, whereas *secondary* resources are shared only among a subset of the jobs but have to be acquired throughout a job's entire processing time. The objective is to obtain a feasible solution that satisfies all resource constraints and minimizes the makespan.

In their work, Horn et al. proposed a complete optimization approach and showed its effectiveness to obtain globally optimal solutions for a diverse set of even large instances. However, some instances turned out to be more challenging for their approach, thus left some room for heuristic improvement. To this end, this thesis proposes a Variable Neighborhood Search (VNS) for the considered problem and experimentally evaluates various aspects of it.

We start this work with efficient evaluation schemes for different neighborhood structures on an encoded solution representation and experimentally show its constant temporal behavior with respect to the number of jobs. We then analyze different aspects of the encountered search landscapes in various experiments and subsequently use the findings to devise a proper VNS, where efficient intensification phases are combined with exponentially more perturbative diversification techniques. Finally, in a series of experiments we study different variations of the devised algorithm, compare them to the selected baseline algorithms from Horn et al. and thereby show its effectiveness to obtain high-quality solutions with an average optimality gap $\leq 0.288\%$ throughout the considered instance classes.

# Kurzfassung

Diese Arbeit behandelt heuristische Verfahren zur Lösung eines Optimierungsproblem, welches in modernen Krebsbehandlungseinrichtungen auftritt, in welchen ein Partikelstrahl mehrere Behandlungsräume abwechselnd bedient. Obwohl in der näheren Vergangenheit bereits verschiedenste Optimierungsprobleme im Kontext solcher Einrichtungen intensiv untersucht wurden, liegt der Fokus dieser Arbeit jedoch ausschließlich auf dem kürzlich von Horn et al. beschriebenen Teilproblem zur Ermittlung von täglichen Ablaufplänen. Insbesondere sind wir daran interessiert die zeitliche Abfolge einer Menge von nicht unterbrechbaren Aufgaben zu bestimmen und dabei eine Zielfunktion zu minimieren. Während der Ausführung der Aufgaben sind grundsätzlich zweierlei Ressourcen anzufordern. Eine Ressource welche von allen Aufgaben geteilt wird, jedoch nur für einen Bruchteil der gesamten Ausführungszeit einer Aufgabe anzufordern ist, sowie einer sekundären Ressource, welche zwar für die gesamte Ausführungszeit benötigt wird, jedoch nur mit einer Teilmenge der anderen Aufgaben geteilt wird. Ziel ist es, eine gültige Lösung zu finden, welche einerseits alle Einschränkungen hinsichtlich der Ressourcen erfüllt, sowie den Zeitpunkt der Beendigung der letzten Aufgabe, die *Makespan*, minimiert.

In ihrer Arbeit stellten Horn et al. ein exaktes Optimierungsverfahren vor und zeigten dessen Effektivität zur Ermittlung optimaler Lösungen für eine Vielzahl an großer Instanzen. Da sich jedoch einige Instanzen für ihren Ansatz als schwieriger herausstellten und heuristisches Verbesserungspotential vermuten ließen, wurde als alternativer Lösungsansatz im Zuge dieser Arbeit ein metaheuristisches Verfahren basierend auf einer Variablen Nachbarschaftssuche (VNS) entwickelt und analysiert.

Hierzu beschäftigt sich diese Arbeit zu Beginn vorranging mit effizienten Evaluierungsansätzen in der kodierten Lösungsrepräsentation und zeigt ein konstantes Laufzeitverhalten bezüglich der Anzahl der Aufgaben in experimenteller Weise. In Folge dessen werden verschiedenste Aspekte von Suchlandschaften in mehreren Experimenten untersucht und dienen als Grundlage für die Entwicklung der VNS, in welcher effiziente Intensivierungsphasen mit exponentiell stärker diversifizierenden Mechanismen kombiniert zum Einsatz kommen. Abschließend werden in einer Reihe von Experimenten verschiedenste Variationen des entwickelten Algorithmus analysiert und mit den Ansätzen von Horn et al. verglichen. Es kann gezeigt werden, dass das entwickelte Verfahren für eine weites Spektrum an Eingabeinstanzen hochqualitative Lösungen mit einem durchschnittlichen *Optimality Gap* $\leq 0.288\%$ finden kann.

# Contents

**Abstract** **ix**

**Kurzfassung** **xi**

**Contents** **xiii**

**1 Introduction** **1**
1.1 Motivation . . . 1
1.2 Problem Description . . . 2
1.3 Aim of this Work . . . 3
1.4 Methodological Approach . . . 3
1.5 Structure of this Work . . . 4

**2 Related Work** **5**

**3 Heuristic Optimization Techniques** **11**
3.1 Local Search . . . 13
3.2 Metaheuristic Optimization Techniques . . . 15
3.3 Variable Neighborhood Search . . . 18

**4 Search Space Structure and Landscape Analysis** **23**
4.1 Search Landscapes . . . 23
4.2 Distance Metrics . . . 25
4.3 Landscape Characteristics and Measures . . . 26

**5 Variable Neighborhood Search for the JSOCMSR Problem** **29**
5.1 Solution Representation . . . 29
5.2 Evaluation . . . 30
5.3 Neighborhoods . . . 34
5.4 Reduction of Neighborhoods . . . 37
5.5 Initial Solution . . . 42
5.6 Shaking . . . 43
5.7 Variable Neighborhood Search for the JSOCMSR . . . 48

**6 Experimental Setup and Fundamental Experiments** **51**
6.1 Environment . . . 51
6.2 Instances . . . 52
6.3 Performance Measures . . . 52
6.4 Synchronization . . . 52
6.5 Evaluation Performance . . . 54

**7 Search Space Structure and Landscape Analysis for Instances of the JSOCMSR Problem** **57**
7.1 Ruggedness of the Landscape . . . 58
7.2 Characteristics of Local Optima . . . 62
7.3 Characteristics of Global Optima . . . 66
7.4 Summary . . . 72

**8 Computational Results** **75**
8.1 GVNS with Randomized Shaking . . . 75
8.2 GVNS with Piped VND . . . 82
8.3 GVNS with Intensified Shaking . . . 85
8.4 GVNS and LLBH . . . 88
8.5 GVNS with Range Limited Neighborhood VND . . . 91
8.6 Neighborhood Pruning based on the Critical Set . . . 96
8.7 Comparison . . . 97
8.8 Summary . . . 99

**9 Conclusion** **101**

**List of Figures** **103**

**List of Tables** **105**

**List of Algorithms** **107**

**Acronyms** **109**

**Bibliography** **111**

CHAPTER 1

# Introduction

In this work, we are interested in heuristically solving instances of the Job Sequencing with One Common and Multiple Secondary Resources (JSOCMSR) problem. This combinatorial optimization problem was first introduced by Horn et al. [HRB17] in the context of patient scheduling in cancer treatment facilities. In this chapter, we start with the motivation for this work, describe the problem in a formal way and outline subsequent chapters.

## 1.1 Motivation

For decades, combinatorial optimization problems have been of interest in the industry and the scientific community. One branch in this research area primarily focuses on scheduling problems, where the goal is among others to assign and arrange a given set of jobs or tasks to a set of machines or resources in such a way that some problem specific cost function is optimized. Scheduling problems are manifold and appear in various aspects of nowadays life, like manufacturing facilities, logistics and health care systems.

One particular case, where the output of optimization-based decision support systems may have a severe impact is in patient scheduling for radio therapy. Patient scheduling is a widely studied topic with numerous variations depending on the particular setting. In the last decades radiation therapy has traditionally been conducted with Linear Accelerators (LINACs) [MRSR16] that are commonly dedicated to a particular treatment room. More recently, a novel technique based on carbon and proton particles became more prevalent, where accelerators (i.e. *cyclotrons* or *synchrotrons* [HRB17]) are shared among treatment rooms in a multiplexed manner due to their more expensive nature in terms of operation cost and space [VBD18].

Generally, the process of a particle therapy and the induced scheduling problem is quite complex, consisting of several irradiations with continuous examinations and imaging

appointments [VBD19], raising the need for tools that support the automated generation of schedules. However, modelling the real-world setting precisely soon turns out to be rather difficult, so often only certain aspects, like the assignment to days in a longer temporal horizon are considered.

The subproblem of finding daily schedules in particle therapy treatment facilities, without considering the entire treatment process was first introduced by Horn et al. [HRB17]. In their work, they studied fundamental characteristics of the JSOCMSR problem, like computational complexity or lower bounds and presented a novel exact optimization approach to find globally optimal solutions. Although their experimental study showed that some instance classes can already be solved to proven optimality, in other classes there is still some room for heuristic improvement.

## 1.2 Problem Description

The JSOCMSR problem consists of a finite set of jobs sharing a common resource that shall be processed without preemption, while minimizing the makespan. In addition to the common resource, each job requires a secondary resource exclusively, which is shared only among a subset of jobs. While the secondary resource is required for the whole processing time, including a setup and removal time, the common resource is needed only for a certain period.

More formally, an instance of the JSOCMSR problem is described by a set of $n$ jobs $J = \{1, \dots, n\}$, a common resource 0 and $m$ secondary resources $R = \{1, \dots, m\}$. Each job $j \in J$ has an overall processing time $p_j > 0$, during which its secondary resource $q_j \in R$ has to be acquired exclusively. At some point during its processing time, each job $j \in J$ has to acquire the common resource for a period of $p_j^0 > 0$ in addition to still being in possession of its corresponding secondary resource. The period between the start of job $j$ and its exclusive acquisition of the common resource, is denoted as its *preprocessing* time $p_j^{\text{pre}} \geq 0$. The postprocessing time of job $j \in J$ starts after the common resource is released and is trivially defined by $p_j^{\text{post}} = p_j - p_j^0 - p_j^{\text{pre}} \geq 0$.

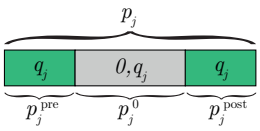



Figure 1.1: Phases of a JSOCMSR job $j \in J$.

A solution $s$ is described by the exact starting times of its jobs $s = [s_j]_{j \in J}$. The makespan of a solution $s$ is then defined as the point in time the last job finishes its execution, i.e. $\text{MS}(s) = \max_{j \in J}(s_j + p_j)$. An example solution of an instance with 20 jobs, three secondary resources and a makespan of approximately 18500 is illustrated in Figure 1.2.

While the problem may be encountered in several manufacturing scenarios as well, it initially appeared in patient scheduling for particle therapy in cancer treatment

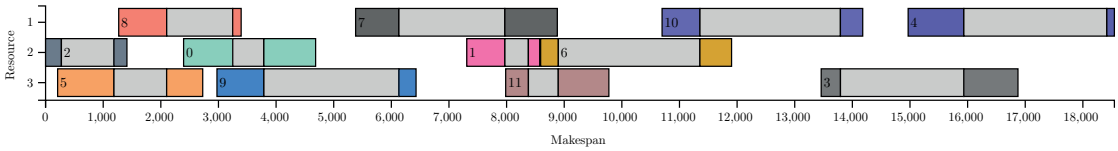



Figure 1.2: Sample solution of an instance of the JSOCMSR problem with $n = 20$ and $m = 3$ secondary resources.

centers. In the process of obtaining a daily utilization schedule, patients are assigned to differently equipped treatment rooms and undergo different preprocessing steps like fixation, alignment, instructing or even sedation, until the particle beam is available for the actual treatment. In practice, the actual irradiation usually only takes up to ten minutes [VBD18], until the irradiation has finished and the particle beam can be released again. Afterwards, patients are usually subject to further medical inspections, until they eventually can leave the room.

## 1.3 Aim of this Work

In their work, Horn et al. [HRB17, HRB19] devised a novel anytime A* algorithm to solve different instances with up to 2000 jobs almost to proven optimality. They primarily focused on two different classes of artificially generated instances. *Balanced* instances, on the one hand, are generated in a way to distribute the jobs uniformly among the available secondary resources. In their study, it turned out that almost all balanced instances of the considered benchmark set could be solved to proven optimality by the suggested A* algorithm. On the other hand, however, instances with a *skewed* workload showed to be more difficult to solve, sometimes even for instances of moderate size.

In the last decades, Variable Neighborhood Search (VNS), a methodology to heuristically improve solutions for optimization problems, showed its effectiveness for many real-world and academical problems [HMM10]. To this end, the aim of this work is to devise alternative solution methods based on VNS for the JSOCMSR problem. The focus lies in particular on the skewed input instances, since they showed to be among the most challenging input instances to solve. Moreover, characteristics of instances are investigated in a landscape analysis to provide insights on the hardness of different instance classes.

## 1.4 Methodological Approach

In this work, we build up on the recent findings of Horn et al. [HRB19] on solution methods for the JSOCMSR problem and investigate alternative, heuristic solution methods. To this end, a VNS is developed with the goal of being able to obtain nearly optimal solutions for a wide range of large instances. As the JSOCMSR is a fairly new problem, we start with a review of more recent literature on radio therapy scheduling and various related

problems, particularly in the context of sequencing and job scheduling. Based on our findings in the literature on recent advances and ideas in the area of VNS algorithms, we then primarily turn our focus on devising and designing a quite generally applicable and efficient VNS algorithm for the JSOCMSR problem.

In the beginning, fundamental concepts, like solution representation, efficient solution evaluation schemes and neighborhood structures are discussed. As it turned out that some components of the algorithm have limitations as instances increase in size, approaches to reduce the size of those neighborhoods are presented, followed by different approaches to perform a systematic diversification in the devised VNS.

Although it is widely acknowledged that metaheuristics are capable of state-of-the-art results for various optimization problems, their internal behavior and more importantly the reason why some approaches yield exceptionally good results is sometimes not totally clear. One approach to shed at least some light onto this uncertainty is to analyze the search space or more precisely the search landscapes induced by the employed neighborhood structures. To this end, before the actual comparison to baseline algorithms is conducted, a landscape analysis based on standard approaches from the literature is performed, to both provide some insights on the structure of the instances as well as on encountered search spaces.

Finally, the devised algorithm and variations thereof are then analyzed in a computational study and compared to the novel anytime A* algorithm proposed by Horn et al. [HRB19]. The algorithms are compared with respect to different instance classes, sizes, number of secondary resources and their temporal behavior.

## 1.5 Structure of this Work

In Chapter 2, we start with an overview of related work in context of particle therapy patient scheduling, but also related scheduling or sequencing problems.

Chapter 3 then provides a brief overview of combinatorial optimization, fundamental concepts of metaheuristics and particularly VNS, followed by a rather compact overview of some landscape analysis techniques in Chapter 4.

The following chapters then present the primary contribution of this thesis. In Chapter 5 the devised VNS is presented, consisting of details about solution representation and evaluation schemes, used neighborhoods and several other aspects. These concepts are then used in some fundamental benchmarks on practical evaluation performance in Chapter 6 and the basic landscape analysis presented in Chapter 7, where properties of the landscapes, like *ruggedness*, *optima depth* and other characteristics are analyzed.

The results of a computational study comparing the devised algorithms with respect to baseline algorithms of [HRB19] are presented in Chapter 8. In this chapter, the outcome of different experiments are presented, where among other things quality and temporal characteristics of the algorithms are investigated. Finally, Chapter 9 concludes this work and presents some ideas for future work.

# CHAPTER 2

# Related Work

Scheduling and sequencing problems have been of great interest for a long time both in academia and industry. Throughout the years an uncountable number of scheduling problems have been studied, with different characteristics, e.g. with respect to the objective function, number of machines or resources, relationship among the jobs to be scheduled or even a job structure similar to the JSOCMSR problem [Pin08].

The JSOCMSR problem was first formally defined by Horn et al. [HRB17] in the context of patient scheduling for radio therapy, where they concentrated on determining exact starting times of jobs of daily schedules. In their work, Horn et al. showed the NP-hardness of the JSOCMSR by a polynomial reduction from the well-known NP-complete *Partitioning Problem* and presented an iterative approach to obtain tight lower-bounds for the makespan of a given instance. Based on this lower bound, they proposed a greedy construction heuristic as well as a novel A* algorithm incorporating a recurring diving mechanism, which both showed their effectiveness compared to a MILP formulation solved by *ILOG CPLEX*. In subsequent work [HRB19], the lower bound was further improved and their A* algorithm was extended by a beam search subprocedure to obtain nearly optimal solutions for previously intractable instances. Since the MILP formulation showed to be not very efficient, a constraint programming model, solved by *ILOG CP Optimizer*, was used as a baseline instead. They considered different classes of input instances, with up to five secondary resources, 2000 jobs and differences with respect to the distribution of the workload to secondary resources. For real-world instances and the ones with a balanced workload, their experimental results showed that the *ILOG CP Optimizer* is hard to beat. In particular, the balanced instances with up to 2000 jobs and only two secondary resources were rather easy to solve, even for the plain constructive approach. However, with increasing secondary resources, the performance of constraint programming model and the construction heuristic declined slightly, while the A* algorithm still managed to solve almost any instance to proven optimality. For the skewed instances, on the other hand, the experimental results drew a rather different

picture. There, instances with only two secondary resources tend to be among the hardest, with a significantly dropping number of obtained globally optimal solutions. Nevertheless, throughout all skewed instances it can be observed that the A* algorithm tends to be significantly more effective in terms of solution quality and temporal behavior than any of its competitors, achieving an optimality gap $< 1\%$ for all considered instances, often in just a fraction of their runtime.

More recently, Horn et al. presented another variation of their sequencing problem. Instead of optimizing the makespan, they aimed to find a feasible solution, while maximizing the sum of the prices associated with jobs, earned when being scheduled within their respective time window [HRR19]. In addition to a variation of their A* algorithm, they presented MILP and constraint programming models for comparison reasons. More recently, approaches based on multivalued decision diagrams and generalized variable neighborhood search [MR19] have been presented as well. In their GVNS, they rely on effective *exchange* and *insertion* neighborhood structures in the improvement phase, while applying random insertions in the shaking phase.

As Horn et al. mentioned, the problem initially originates in scheduling of patients in an Austrian particle therapy treatment center [HRB17][1]. Rare resources compared to the steadily increasing demand in nowadays health care systems make it apparent that scheduling and optimized resource provisioning becomes a more and more important aspect. Hence, applying optimization techniques to utilize valuable resources more efficiently is not a novelty and has been studied in the literature for decades, ranging from staff timetabling [EJKS04, CLLR03, KBDCVBVL04] and material logistics [VFSB17], to planning of operating rooms [CDB10] and last but not least various aspects of patient scheduling [CV09, VHvVV+16, MD19].

Scheduling in radiotherapy is a manifold topic induced by the quite diverse landscape of applied techniques, used equipment, processes and constraints in different facilities. In radiotherapy, usually a technique called *fractionation* is applied, where reduced portions of irradiation are distributed among several consecutive days to allow healthy surroundings to recover [PL08]. Depending on the type and progress of the disease, the intensity, duration of the treatment and the required equipment (e.g. energy level of the irradiation) may vary from patient to patient. Additional constraints like appointments for imaging activities or checkups at physicians induce a highly constrained problem with several potential optimization objectives.

Employing optimization techniques in radiotherapy patient scheduling seems to appear initially in the early 1990's, where Larsson [Lar93] describes the introduction of a spreadsheet-based system, where move operations have been implemented as macros to reduce waiting list lengths. Later on, the problem of patient scheduling in radiotherapy was defined more precisely and more advanced optimization techniques have been studied. In the beginnings, research in this area primarily focused on patient scheduling with LINACs, where typically an accelerator is dedicated to a particular treatment room

[1] http://medaustron.at/

and patients are assigned to treatment rooms based on personalized treatment plans, priorities and resource capacities. Petrovic et al. [PLSS06] first studied constructive approaches based on iteratively applied dispatching rules to minimize tardiness with respect to waiting time targets. In their work, they deal with the problem of assigning patients to treatment slots on a daily basis, while also considering constraints with respect to equipment, treatment duration and priorities. In subsequent work, they presented improved constructive and metaheuristic approaches based on greedy adaptive search procedure (GRASP) [PL08] and steepest hill climbing [KP09]. Later on, more metaheuristic approaches have been studied for their problem or variations thereof, primarily based on genetic algorithms [PMP09] or hybrid optimization techniques [CP12].

More recently, alternatives to classical photon radiotherapy became more prevalent, where irradiation is based on ion beams to minimize damage of healthy surroundings. However, since these accelerators are usually more expensive in terms of cost and space, they commonly serve multiple treatment rooms in a multiplexed manner [MRSR16, VBD18]. In practice, the actual irradiation in such treatments centers usually takes up to ten minutes, whereas preparation and post-processing steps, like orientation and alignment or instructing the patient may take a significant amount of time as well. Serving available rooms in an interleaved manner allows to already utilize the ion beam, while other patients are still subject to their respective pre- and post-processing phases.

In the last years, the problem of scheduling patients in ion beam facilities was intensively studied in the literature. The problem was initially introduced by Maschler et al. [MRSR16], dealing with patient scheduling in an Austrian radiotherapy treatment center. In contrast to previous work in patient scheduling in radiotherapy, the novel irradiation technique not only requires assigning patients to treatment timeslots on a daily basis for a given time horizon, but also an efficient resource utilization in daily schedules. Maschler et al. [MRSR16] started with constructive approaches incorporating forward-looking mechanisms for improved greedy decisions. The constructive approaches were then reused in GRASP and iterated greedy (IG) metaheuristics to further improve the solution quality [MRSR16, MHRR17]. In their first works, they deal with a relaxed version of the problem, where the daily assignment is independent of sequencing the patients each day. However, in practice this assumption does not hold always true and patients prefer having their daily treatments approximately at the same time. In a more recent work, Maschler et al. [MR18] incorporated this aspect into their previous IG approach, by changing the construction heuristic and incorporating a linear programming (LP) model into the local improvement phase.

For a similar problem, Vogl et al. [VBD18] proposed a genetic algorithm to minimize both time-windows violations as well as idle times of the ion beam. In their genetic algorithm, they make use of a combination of assignment and permutation encodings to capture different aspects of the problem and incorporate a decoding algorithm to derive the actual schedule with exact job starting times. In their first work, they mainly rely on state-of-the-art crossover operators, like position-based crossover (PBX) or approaches where children inherit the entire *genotype* from one parent. Later on, they presented

a feasibility preserving crossover operator and introduced an Iterated Local Search based on a variable neighborhood descent with several sequentially applied neighborhood structures [VBD19].

In contrast to the work by Maschler and Vogl that primarily focus on a concrete patient scheduling problem, Horn et al. [HRB17] mentioned that the JSOCMSR is not limited to patient scheduling and may also appear in sequencing problems or flow-shop scenarios as they are frequently encountered in ordinary production lines.

Sequencing problems have been of major interest since the early beginnings in scheduling research [All15]. However, problems with multiple resources and a job structure consisting of pre- and postprocessing phases are rarely found in the literature [HRB17]. Despite this, a closely related problem, where post-processing times are assumed to be negligible and jobs are subject to preprocessing phases before the common resource is utilized, has been studied extensively in the last decades. Primarily introduced already in the 1960's, the problem and numerous variations thereof have been of major interest in the industry [All15]. Gilmore and Gomory [CGG64] first dealt with a sequencing problem where jobs consisting of an additional setup time, have to be scheduled on a single machine, while minimizing transition costs between subsequent jobs. By formulating the problem as specially structured traveling salesman problem (TSP), they show that the optimal sequence can be found in $\mathcal{O}(n^2)$ steps. Later on, Veen et al. [vdVWZ98] considered a similar problem, where the transition costs between subsequent jobs depends on the *template* a job is assigned to. In their work, Veen et al. consider a typical production scenario, where $K$ templates are available for $n$ jobs and the single machine can process only a single job at a time. The *change-over time* between subsequent jobs then depends on whether the jobs make use of the very same or a different template. In their work, they also model this problem as a specially structured TSP and provide a $\mathcal{O}(n \cdot \log n)$ algorithm for the optimal sequence. A similar problem has been studied in the context of job scheduling on parallel machines, where a single machine performs the preparation of a job, until the job is finally executed on one of a set of parallel machines. Hall et al. [HPS00] first studied polynomial and pseudo-polynomial algorithms to optimize different objectives, like makespan, lateness, total completion time and some more.

Although for some of the previous approaches a specially structured TSP allowed to find the optimal sequence in polynomial time, this apparently does not hold true for all variations of sequencing problems. Glass et al. [GSS00] considered a makespan minimization problem similar to Hall et al. [HPS00], where during the setup phase, both the common and the primary resource must be available. In their work, they prove the NP-hardness in the strong sense and provide a greedy algorithm that already guarantee rather tight worst-case approximation ratios. Nowicki and Zdrazalka [NZ96], considered an NP-hard single machine sequencing problem, where jobs are assigned to families that affect the change-over time between subsequent jobs. While for jobs in the same family the setup time is minor, a transition between jobs of different families is much more expensive. For their problem, they propose a tabu search (TS) algorithm to find

nearly optimal solutions with respect to maximum weighted lateness and total weighted tardiness objectives. In their tabu search, they mainly rely on insertion moves that incorporate mechanisms to reduce the cardinality of the neighborhood and approaches to evaluate the neighborhoods more efficiently. In the literature, even more variations of such sequencing problem can be found that share some characteristics, like sequence independent setup costs, with the JSOCMSR. A comprehensive survey can be found in [All15].

Whereas the JSOCMSR is indeed a fairly new problem some aspects of the problem are frequently encountered in flow-shop or hybrid shop scheduling problems. The no-wait job structure during the transition between secondary and common resources, is fairly common in flow shop problems, where the processing of a job by subsequent resources has to start immediately after the job has been release by former resources. A comprehensive survey is given by Allahverdi [All16].

As mentioned previously, scheduling problems dealing with setup phases are quite common, whereas post-processing phases are rarely seen. However, they have been considered in some work in the context of flow shops. Yoshida and Hitomi [YH79] first considered a two-machine flow shop problem with dedicated setup times. Later on, Sule and Huang extended this model to job dependent setup and removal times for two [Sul82] and three [SH83] machines. More recently, Chang et al. [JWH04] considered a no-wait hybrid flow shop problem with setup and removal times, consisting of two stages, with one and $n$ machines respectively.

Another, to some extent similar problem has been studied by Agnetis et al. [AFNP11]. In their work, they deal with a resource constraint job scheduling problem, where in addition to the $m$ available machines, each job has to acquire an additional resource with limited availability $p < n$. They showed the NP-hardness of the problem already for $n = 3$ and $p = 2$ and proposed pseudo-polynomial as well as approximate algorithms for their problem.

Finally, Horn et al. [HRB17] state that the JSOCMSR can also be reduced to an instance of the Resource-Constrained Project Scheduling Problem (RCPSP) with maximum time lags. In the literature, the PCPSP is known as a standard scheduling problem, where a set of *activities* (i.e. jobs), which may be subject to precedence constraints are to be scheduled on a set of resources [HB10] to minimize the makespan. Maximum time lags specify the maximum delay between the end of a former and the start of a latter job. When splitting the jobs of the JSOCMSR into three separated tasks subject to precedence constraint with maximum time lag of 0 and designing the resource constraints respectively, it becomes apparent that each instance of the JSOCMSR can be reduced to an instance of the RCPSP with maximum time lags. While this transformation is indeed straight forward, it is important to note that increased instance size by a factor 3 and the relatively high number of $2n$ precedence constraints, make such a reduction a rather inefficient approach for obtaining good solutions for non-trivial instances of the JSOCMSR.

CHAPTER 3

# Heuristic Optimization Techniques

For decades, optimization problems have been of major interest in the industry and the scientific community, due to their practical relevance in various disciplines and their challenging nature. In practice, optimization problems arise in various areas, ranging from allocation and scheduling problems, up to topics in hardware design and genome sequencing [HS04]. In the literature, optimization problems are usually divided into continuous and combinatorial problems. While in *continuous* optimization problems, the decision variables are usually subject to real-valued numbers, the search space in *combinatorial* or discrete optimization problems consists of a finite set of possible solutions [HPS82].

---

**Definition 3.1 (Combinatorial Optimization Problem [Tal09]):**
*A combinatorial optimization problem* $\Pi$ *is defined by a pair* $\Pi = (\mathcal{S}, f)$*, where the search space* $S$ *being a finite set of candidate solutions (often referred to as solution space as well) and* $f$ *an objective function that assigns a real-valued objective value to each solution* $s \in \mathcal{S}$*.*

*While in* minimization *problems, the goal is to find a globally optimal solution* $s^\star \in \mathcal{S}$*, s.t.* $\forall s \in \mathcal{S} : f(s^\star) \leq f(s)$*, for a globally optimal solution* $s^\dagger \in \mathcal{S}$ *in a* maximization *problem,* $\forall s \in \mathcal{S} : f(s^\dagger) \geq f(s)$ *holds true* [1]*.*

---

[1] In the following we primarily focus on minimization problems, however it should be pointed out that a conversion to maximization problems is trivial and thus the presented concepts apply to both minimization as well as maximization problems.

Since the early beginnings in the 1940s, research on optimization problems attracted a lot of attention in the scientific community. Since then, various research branches have emerged under the heading of *Combinatorial Optimization* that focus on different aspects and various kinds of optimization problems.

For a given optimization problem $(\mathcal{S}, f)$, approaches usually aim to explore the search space $\mathcal{S}$, in order to find one out of potentially enormous number of global optimal solutions $s^\star \in \mathcal{S}$. At first glance, this seems like an easy task, since an enumeration of the entire search space $\mathcal{S}$ will eventually find the optimum. However, practice has shown that a majority of the combinatorial problems turned out to be $\mathcal{NP}$-hard with a solution space growing exponentially with the size of the input [Tal09]. As a consequence, applying a naive brute-forcing algorithm for such problems will become infeasible already for instances of almost trivial size and hence more sophisticated approaches are required and have actually been devised and continuously refined throughout the last years.

One branch of research primarily focuses on *exact* approaches, which are often based on common principles like *divide-and-conquer*. They try to systematically explore the entire search space $\mathcal{S}$, while explicitly avoiding regions where the optimum is certainly not to be found. The advantage of those approaches is indeed their completeness, i.e. they yield a proven optimal solution on termination. However, increasingly complex problems and large instance sizes usually turn out to be a major limiting factor for almost any of them. Prominent exact approaches in the context of optimization are certainly *mathematical programming techniques*, algorithms based on principles like branch-and-bound or dynamic programming or last but not least approaches based on informed search algorithms (e.g., A*-family) [Tal09].

Fortunately, in some scenarios, nearly optimal solutions suffice, which allows the use of *heuristics*. Originating from the Greek term *heuriskein* for finding or discovering [SG13], heuristics are approaches that aim to obtain good solutions at reasonable computational cost. However, they inevitably come with the trade-off of being *incomplete*, i.e. in contrast to exact approaches, they usually do not provide any guarantees with respect to solution qualities and worst-case execution time[2], but rather focus their search only on particular regions in the search space and therefore may miss out promising regions potentially containing globally optimal solutions.

Under the umbrella term heuristic, a wide spectrum of different approaches is subsumed, ranging from constructive approaches that stepwise build a new solution from scratch, to improvement heuristics that modify an initial solution in a strategic way to obtain new solutions, thereby exploring promising regions of the search space. Neighborhood search, a very prominent class of these approaches, is based on the idea that a solution can be improved by investigating its direct surroundings with respect to a *neighborhood structure*, i.e. the *neighborhood*, and follow a path of solutions in the search space towards improved solution quality according to some predefined rules.

[2]It is important to note that for some metaheuristics like *Simulated Annealing* [AL97] or *Ant-Colony Optimization* [DB05] indeed bounds with respect to their convergence behavior exist. However, often these bounds only apply in some special cases and are hence only of limited practical use.

**Definition 3.2 (Neighborhood [BR03]):**
*For a combinatorial optimization problem $\Pi = (\mathcal{S}, f)$, a neighborhood structure is a function $\mathcal{N} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ that defines for each solution $s \in \mathcal{S}$ a set of solutions $\mathcal{N}(s) \subseteq \mathcal{S}$ that constitute the neighborhood of $s$*[3].

Often, neighborhood structures are based on small perturbations or moves applied to a solution, which is the reason why we subsequently sometimes use the terms synonymously.

The structure of the solution space, however, is usually not really smooth and may contain misleading structural elements, i.e. the local optima, that prevent the search from any further progress of reaching the optimal solution.

**Definition 3.3 (Local Optima [BR03]):**
*For a combinatorial minimization problem $\Pi = (\mathcal{S}, f)$, a solution $s^\star$ is locally optimal with respect to a neighborhood structure $\mathcal{N}$, if $\forall s \in \mathcal{N}(s^\star) : f(s^\star) \leq f(s)$. Note that each global optimum is also locally optimal with respect to any neighborhood structure.*

## 3.1 Local Search

One of the central and most fundamental heuristics is certainly the *local search*. Local search algorithms usually start with an initial solution and iteratively explore small neighboring regions in the search space to identify solutions of improved quality. By following a path of neighboring solutions iteratively in direction of improved solution quality, the search converges eventually towards a local optimum.

In the literature, local search procedures seem to initially appear in the late 1950s [HPS82], in the context of the TSP. In their work, Croes [Cro58] iteratively applied transformations — called *inversions* — to an initial solution to reduce the cost of the tour. Later on, Lin [Lin65] and Reiter [RS65] enhanced this approach by applying more complex transformations that improved the generated local optima even more.

As can be seen in Algorithm 3.1, a local search procedure is characterized by three main components: the solution representation, the neighborhood structure and the search strategy. In the following, the three components will be described in more depth.

**Solution Representation** One of the basic ingredients of a local search procedure for an optimization problem is the solution representation. As such, it has a crucial impact on both the efficiency as well as the effectiveness of the search. While compact representations may improve efficiency in terms of memory and cache utilization, more

[3] For a given set $S$, the powerset of $S$ is denoted by $\mathcal{P}(\mathcal{S})$

**Algorithm 3.1:** Local-Search

**Input:** An initial solution $s_0$, a neighborhood structure $\mathcal{N}$, a search strategy $\mathcal{T}$.
**Output:** A locally optimal solution s.

```
1 s ← s_0;
2 repeat
    // select successor from neighborhood based on a search strategy
3   s' ← select(N(s), T);
4   if f(s') < f(s) then
5     s ← s';
6   end
7 until s is locally optimal or another stopping criterion is satisfied;
8 return s;
```

complex representations and data structures may be beneficial for faster evaluating the objective value of a solution. Apparently, an appropriate solution representation depends on the problem at hand. While for assignment and partitioning problems usually decision variables with certain domains are used, scheduling and routing problems often make use of linear or cyclic permutation representations.

**Neighborhood Structure** Commonly, neighborhood structures are based on move operators that are applied to a subset of solution components. While the move operator has to be chosen in accordance with the problem at hand, a difficult trade-off has to be taken with respect to the size of the neighborhood. Whereas for rather small neighborhoods (e.g. $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$) the evaluation is usually rather efficient, for some problems they tend to converge towards local optima quite fast. On the other hand, local optima of larger neighborhoods usually tend to be of higher quality. However, the higher computational effort for evaluating the neighborhood usually turns out to be a major bottleneck in the search. Already in the very beginning of research in this area, researchers came up with approaches to overcome this problem with variable sized neighborhoods [LK73]. Throughout the years various other techniques have been proposed. In particular, large neighborhood search (LNS) [TO$^+$89, AOS00], VNS [Mla95] and dynasearch [Con00] are amongst the most prominent ones.

**Search Strategy** The search strategy usually decides, which solution in the neighborhood is selected as successor. Strategies that always follow the steepest descent by scanning the whole neighborhood are usually referred to as *best improvement*, whereas *first improvement* methods follow the path towards the first improving neighbor. Alternatively, some extensions to local search, like Simulated Annealing, make use of a different search strategy, namely *randomized descent*. In contrast to the previous strategies, this approach does not evaluate the neighborhood systematically, it rather samples a potential successor randomly.

In general, it is not totally clear, whether one search strategy is actually superior to the others and hence no general statement can be made. Indeed, the structure of the solution space plays an important role, but also other factors like the applied neighborhood structure or the construction of the initial solutions have a non-negligible impact [HM06].

**Limitations** For some optimization problems, in particular when the solution space is *convex*, local optima can already produce satisfying results. However, for more complex problems, the solution space is often rich in low-quality local optima and hence inevitably leads to a rather fast termination and traps the search in unsatisfying regions. Since this is apparently a significant limitations of local search heuristics, throughout the years numerous approaches have been presented with the sole purpose of detecting local optima and diversify the search to allow the procedure to explore more promising regions in the search space. Nowadays they are usually subsumed under the more general term *metaheuristics*.

## 3.2 Metaheuristic Optimization Techniques

In the last centuries, metaheuristics have been established as viable alternative to exact approaches for a wide range of hard optimization problems.

By incorporating problem-specific heuristics and operations into a problem-independent, high-level framework of search strategies, metaheuristics guide the search through the solution space and often allow to identify promising regions with high quality solutions in an efficient way. To cope with search spaces exponentially in the size of the input instance, metaheuristics are often of stochastic nature and employ learning and memorization mechanisms to improve the efficiency of the search space exploration.

Throughout the years, a variety of different approaches have been proposed to cope with common pitfalls, like premature convergence, lack in diversity or cycles. On the one side of the spectrum reside trajectory-based methods, ranging from simple local search extensions, up to advanced mechanisms with variable or very large neighborhoods. On the other side, population-based techniques are driven by information gathered through a set of solutions and often mimic natural phenomena or stigmergy. In the following sections, we will briefly introduce commonly used metaheuristic search techniques.

### 3.2.1 Trajectory Methods

Trajectory-based methods are essentially based on the local search heuristic, where modifications are iteratively applied to incumbent solutions in order to improve the objective value. While most techniques still use some variant of the traditional local search in their improvement phase, they mainly differ in the way, how local optima are overcome and the search is diversified.

One of the first, widely accepted mechanism to escape from local optima was introduced by Kirkpatrick [KGV83] and Černý [Čer85] under the term *simulated annealing*. By

adopting the concept of annealing from thermodynamics to combinatorial optimization, an innovative method was introduced, which showed to be very effective for numerous optimization problems. Starting from an arbitrary initial solution and appropriate values for the algorithm's parameters, most importantly the *temperature*, the search simulates the process of gradually cooling solids until a state of *thermal equilibrium* is achieved. Essentially, a random-neighbor local search is performed, that accepts possible deteriorating solutions, in case the temperature-dependent *Metropolis Criterion* [MRR+53] is met. During the search process, the temperature is then gradually decreased, enabling diversification in the beginning of the search and intensification as the temperature approaches its minimum. Apparently, the selection of parameters like the initial temperature or the cooling schedule are crucial and highly affect the performance as well as the convergence behavior of the search. Various endeavors have been made to adapt the basic scheme of simulated annealing by introducing nonmonotonic [HKT95] or adaptive [Ing96] cooling schedules and many other extensions to improve both performance and convergence behavior.

In contrast to simulated annealing, other endeavors in the community tried to incorporate information of previous iterations to guide the search towards promising regions. Whereas many population-based techniques rely on implicit information contained in individuals, one of the most prevalent trajectory-based metaheuristic, the tabu search (TS), explicitly makes use of memories to keep track of characteristics of previously visited solutions or attributes. First introduced by Glover [Glo86] and Hansen [Han86], TS is in its essence a rather deterministic extension of ordinary steepest descent local search procedures, employing memorization mechanisms to guide the search in intensification and diversification phases. To escape from local optima, tabu search usually follow trails of degrading objective, while avoiding cycles with so-called short-term memories. To avoid that the search gets trapped in only a few basins of attraction in the long term, memories with longer temporal scopes are usually incorporated to guide the trajectory towards more promising areas. Again, throughout the years a variety of extensions, like adaptive size of memories [NI98] or advanced diversification mechanisms [GP10, GH11], have been proposed.

Next to the most prominent members of trajectory-based approaches with *simulated annealing* and TS, throughout the years a wide range of other techniques have been introduced. While some approaches, in particular GRASP [FR89] and its ancestors [HS87], behave similarly to randomized construction heuristics in combination with local optimizers, various alternatives are commonly used. Some approaches, like iterated local search (ILS) [LOT10] are based on destroy-and-repair principles, other prominent techniques make use of dynamic or adaptive mechanisms, like changing neighborhood structures or objective functions to handle complex structures in the search landscapes [Tal09]. One of the most successful techniques in this category is certainly VNS [Mla95]. As VNS is the primary topic of this work, a more elaborated introduction to basic principles and viable extensions is given in Section 3.3.

### 3.2.2 Population based Methods

Population-based methods are usually based on the concept of iteratively adapting and replacing individuals in a set of solutions, based on information obtained in previous iterations. Commonly, they are based on biological analogies or at least inspired by natural phenomena and intelligent behavior like stigmergy. Since early pioneering works have emerged in the early 1960s, journals and conferences have been flooded by various approaches.

One of the most influential classes in the field are certainly evolutionary algorithms. Initially, they appear in the context of continuous optimization [Rec70], but similar approaches have been applied to combinatorial problems soon afterwards [Hol75]. Based on the basic principles of natural evolution, the field has evolved tremendously, emerging also towards concepts of evolving programs [Koz92] and finite automatons [FOW66]. While the application area of these branches of evolutionary computing is quite diverse, they usually differ only in basic building blocks, like type of a solution and its representation, recombination and mutation operators and selection schemes. Further, all kinds of evolutionary algorithms share major difficulties, like preservation of diversity and premature convergence towards local optima, which is the reason why this field has continuously evolved and various extensions, like hybrid mechanisms [MC10], haven been proposed throughout the years. A comprehensive collection may be found in [ES03].

Next to evolutionary algorithms, swarm intelligence approaches constitute another important group in population-based metaheuristics. Usually adopting intelligent behavior of natural processes, these techniques commonly employ indirectly communicating agents [Tal09] to guide the search or the incremental construction of solutions. Initially driven by a few pioneering works on *particle swarm optimization* [KE95] and *ant colony optimization* [Dor92], the community did not lack in finding new analogies or natural phenomena from which new optimization techniques were derived. Ranging from bees to mosquitoes and from particle swarms to artificial immune systems [Sia16], the field was flooded by new concepts and methodologies. While from this trend a plenty of promising, successful and widely used techniques, like algorithms based on *ant colony optimization* [DD99, DS10], have emerged, critics often miss real innovation in new approaches [SÏ3].

### 3.2.3 Performance of Metaheuristics

As the previous sections have shown, since the early beginnings, the field of heuristic optimization techniques has flourished. Since then, numerous approaches have been proposed to tackle an even larger number of optimization problems and variations thereof. However, the nature of optimization problems is usually rather diverse. While some problems come with exceptionally huge instances, others may have a highly complex search space induced by hard constraints. Apparently, it is not unusual that superior approaches for one problem, perform poorly for others. While this observation was already widely recognized in the field of Genetic Algorithms [Whi13], it was first formalized in the

late 1990s and is now commonly known under the term *No Free Lunch Theorem* [WM97]. In particular, it states that on average there is no algorithm that performs significantly better than any other for all possible performance measures, problems and instances. As a result, for a particular problem at hand, assumptions regarding one superior technique can in general not be made and thorough analysis as well as experimental comparison of different approaches is of highest importance.

In metaheuristics, deciding on a particular metric to quantify the performance of an algorithm is also not apparent. Due to their incomplete nature, several aspects can and should be considered. In optimization, obviously the solution quality plays a central role. However, depending on the application's requirements, other characteristics, like the temporal behavior or robustness against structural changes of instances may also be of significant relevance [Tal09].

## 3.3 Variable Neighborhood Search

Variable Neighborhood Search is a trajectory-based metaheuristic framework, employing sets of neighborhood structures to both improve local optima and regularly diversify the search to explore more promising regions in the search space. Initially, VNS was introduced as a mechanism to guide an embedded local search out of suboptimal regions [Mla95]. As early results showed the effectiveness of this approach, research in this area flourished and various adaptions and extensions have been proposed in the literature [HMTH17]. Ranging from deterministic local optimizers (e.g. Variable Neighborhood Descent), up to multi-level schemes for complex and large instances or other powerful hybrid techniques [HMP01, HMTH17], the family of VNS algorithms has established itself as one of the most successful metaheuristics for a wide range of problems.

In general, VNS is based on the following observations [HMM10]:

(i) Locally optimal solutions with respect to one neighborhood structure may not be (locally) optimal in other neighborhood structures.

(ii) Globally optimal solutions are locally optimal in all possible neighborhood structures.

(iii) For many problems, empirical evidence shows that many elite solutions share important solution components and are hence relatively close together, i.e. local optima are usually not evenly distributed amongst the search space, rather concentrated in some regions.

So based on these observations, VNS makes use of up to two sets of neighborhood structures that are systematically applied in two alternating phases: *intensification* and *shaking*.

**Intensification** The intensification phase primarily aims to thoroughly explore surrounding regions of the incumbent solution and is primarily responsible for obtaining improved solutions throughout the search. To this end, usually local optimization procedures, like local search are employed, which can efficiently identify improving neighbors and eventually the respective local optimum.

**Shaking** The shaking phase is incorporated into the search to escape from local optima the *intensification* phase has been trapped in. To this end, the shaking phase relies on a set of shaking neighborhood structures that apply random perturbations to the incumbent solution to shift the search toward more distant areas. Usually, these neighborhood structures are defined with increasing cardinality to allow the search to explore closer areas around the current solution first. In case these areas turn out as suboptimal, shaking neighborhood structures of higher order are subsequently applied to concentrate the search to more distant areas. Here, special care must be taken to find a good trade-off between preservation of structural components of the incumbent solution and the introduction of significant, randomized perturbations to foster diversity in the search.

### 3.3.1 Variable Neighborhood Search Variants

In principle, the term variable neighborhood search describes a rather general framework of how several neighborhoods can be combined or incorporated to keep to some extent the simplicity of local search procedures, while still having a powerful framework to avoid fast convergence and poor solution qualities. In the literature, several slightly different approaches have been presented that put the focus on different aspects of the search. In the following, all approaches relevant for this work are presented briefly. For more in depth information, we refer to Hansen et al. [HMTH17].

**Variable Neighborhood Descent** Variable Neighborhood Descent is certainly the most basic variant and can be seen as a generalization of the basic local search. In contrast to the ordinary two-phased approach in VNS, variable neighborhood descent (VND) does not incorporate a shaking phase and is therefore an entirely deterministic search procedure. Starting from an initial solution, the first neighborhood structure is applied until a local optimum with respect to the first neighborhood structure is obtained. Subsequent neighborhood structures are then used to repeatedly escape from local optima obtained by previous ones, until eventually a solution being locally optimal in each neighborhood structures is obtained. A pseudocode of a basic, sequential VND can be seen in Algorithm 3.2.

Commonly, VNDs follow a steepest-descent strategy. However, depending on the number and the cardinality of the neighborhoods, other strategies, like first-improvement may be more appropriate. More importantly, however, is the order in which the neighborhood structures are applied. Due to the nested fashion and the ability to escape from one local optimum with subsequent neighborhood structures rather easily, applying large

or computational expensive neighborhoods in early phases may turn out as a major bottleneck.

**Algorithm 3.2:** Variable Neighborhood Descent

**Input:** An initial solution $s_0$, a set of intensification neighborhood structures $\mathcal{N}^I_{i=1\ldots l_{\max}}$ and a search strategy $\mathcal{T}$.
**Output:** A locally optimal solution s.

```
1  s ← s_0;
2  l ← 1;
3  repeat
4  |   s' ← select(N_l^I(s), T);
5  |   if f(s') < f(s) then
6  |   |   s ← s';
7  |   |   l ← 1';
8  |   else
9  |   |   l ← l + 1;
10 |   end
11 until l ≥ l_max or other stopping criterion is satisfied;
12 return s;
```

**Fixed Neighborhood Search** One of the first variants of VNS is the so-called fixed neighborhood search (FNS) [BHMT00, HMTH17]. Instead of using sets of neighborhood structures, FNS makes use of a single perturbation neighborhood as well as a neighborhood for an embedded local search[4]. For problems, where identifying local optima after each shaking phase is too expensive, some variations, like basic variable neighborhood search (BVNS) or reduced variable neighborhood search (RVNS), entirely skip or reduce the intensification phase to a minimum. While these techniques are commonly used for large instances, approaches presented in the following may be used for complex instance and search space structures as well.

**General Variable Neighborhood Search** As the name implies, the so-called general variable neighborhood search (GVNS) given in Algorithm 3.3 is a rather general VNS variant. Here, the local intensification phase consists of a VND subprocedure that may enable the search to obtain high quality local optima. Originally, the order in which the shaking neighborhood structures are applied in Shake$(s, \mathcal{N}^S_k)$ was strictly sequential, starting with $k = 1$ and moving to the next neighborhood structure in case no improved solution could be obtained. Based on this, in the literature alternative strategies have been proposed that change the algorithms behavior how the shaking neighborhood structures, or even intensification neighborhood structures are changed throughout the search. Whereas Di Gaspero et al. [DGS06] propose a so-called *token-ring*

[4]It is important to note that FNS is very similar to ILS [LOT10]. To some extent, VNS could be considered as a generalization of ILS.

or *pipe* approach, where the search sticks to a successful neighborhood structure, until no improvement can be achieved, Todosijevic et al. [TMM+17] introduced the *cyclic* step function that changes the neighborhood structure in each iteration.

**Algorithm 3.3:** General Variable Neighborhood Search

**Input:** An initial solution $s_0$, a set of shaking neighborhood structures $\mathcal{N}^S_{i=1\ldots k_{\max}}$, a set of intensification neighborhood structures $\mathcal{N}^I_{i=1\ldots l_{\max}}$, a change step function Change$(s, s', k)$ and a search strategy $\mathcal{T}$.

**Output:** An improved solution s.

$s \leftarrow s_0$;
**repeat**
  $k \leftarrow 1$;
  **while** $k \leq k_{\max}$ **do**
    $s' \leftarrow \text{Shake}(s, \mathcal{N}^S_k)$;
    $s'' \leftarrow \text{VND}(s', \mathcal{N}^I, \mathcal{T})$;
    $(s, k) \leftarrow \text{Change}(s', s'', k)$;
  **end**
**until** *stopping criterion is satisfied*;
**return** $s$;

CHAPTER 4

# Search Space Structure and Landscape Analysis

The heuristic nature and the lack of theoretical results makes it often hard to understand under what conditions metaheuristics perform well and what problem or instance characteristics amount to the major factors affecting the *hardness* of a problem.

One way to analyze these characteristics is to take a closer look at the search space. However, the search space is formally only defined as a set of solutions (see Definition 3.1) and does not describe any topological structure of its elements nor is it bound to a particular solution representation. To this end, in the literature often the concept of *search landscapes* is used. Search landscapes describe the search space structure induced by a given solution representation and the topological relationship between the solutions due to a particular neighborhood structure.

In the remainder of this chapter, we will start with a more detailed description of *search landscapes* and provide a formal definition. Then, we will direct our focus on the *distances* encountered in such landscapes, due to their importance in landscape analysis techniques. Finally, the chapter will end with a summary of relevant analysis techniques, where some were applied later on in the context of the JSOCMSR.

## 4.1 Search Landscapes

The concept of a *search landscape* initially appeared in the context of evolutionary biology already in the 1930s [Wri32] and it were also similar communities in *Bioinformatics* and *Evolutionary Algorithms* that were the major drivers in the context of combinatorial optimization [HS04, Kau93]. For neighborhood search techniques, basic landscape analysis was initially conducted for well-known combinatorial problems, like *traveling salesman problem* [SS92], *graph coloring* [HJdA94] and the *satisfiability problem of boolean formulas*

(SAT) [FCS97]. Throughout the years, various techniques have been devised to describe the structure of the search space of problems in more depth. In the literature, landscapes are commonly defined on the basis of a vertex-weighted, directed graph [Wat10].

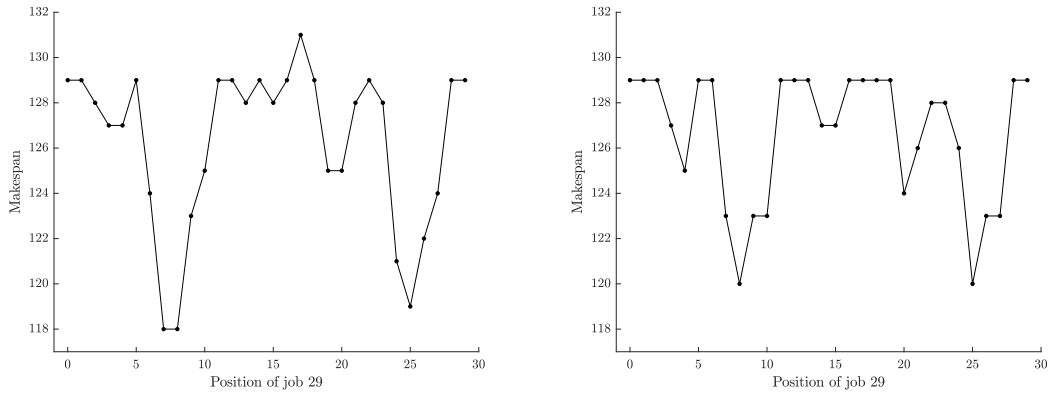
---

**Definition 4.1 (Landscape):**
*The landscape $\mathcal{L}$ of a combinatorial optimization problem $\Pi = (\mathcal{S}, f)$ over the neighborhood structure $\mathcal{N}$ is defined as a vertex-weighted, directed graph $\mathcal{G}_\mathcal{L} = (\mathcal{V}_\mathcal{L}, \mathcal{E}_\mathcal{L}, \omega)$, with a vertex set $\mathcal{V}_\mathcal{L}$ and an arc set $\mathcal{E}_\mathcal{L}$. Furthermore, let $\delta$ be a bijective function $\delta : \mathcal{S} \to \mathcal{V}_\mathcal{L}$ that maps each solution $s \in S$ to exactly one vertex $v \in \mathcal{V}_\mathcal{L}$ and a weight function $\omega : \mathcal{V}_\mathcal{L} \to \mathbb{R}$, where*

1. $\forall s \in \mathcal{S} : \omega(\delta(s)) = f(s)$
2. $\mathcal{E}_\mathcal{L} = \{(i,j) | i \neq j \wedge i \in \mathcal{V}_\mathcal{L} \wedge j \in \mathcal{V}_\mathcal{L} \wedge \delta^{-1}(j) \in \mathcal{N}(\delta^{-1}(i))\}$

---

More informally, the landscape is a structure on the search space induced by the representation and the applied neighborhood structure. This structure is represented by the neighborhood graph, where neighboring solutions are adjacent vertices with weights according to their objective value. While some characteristics of landscapes, like *connectivity* are easily defined over this graph, other significant characteristics, like valleys, plains, peaks, canyons or plateaus [Tal09] may be more intuitively visualized in the 2D or 3D topographic analogy. Figure 4.1, shows illustrative examples of the landscape of a one-dimensional optimization problem, where exchange and insertion moves are applied to an instance of a tiny JSOCMSR subproblem. At first glance, it can be seen that both landscapes show similar patterns with respect to their local optima at position 8 and 25. However, at position 20, this example also illustrates one of the main observations VNS is based on: *Local optimal solutions with respect to one neighborhood structure may not be (locally) optimal for other neighborhood structures.* Furthermore, other significant regions commonly found in landscapes, like benches and plateaus can also be seen in several regions in this example.




(a) Exchange Neighborhood (b) Insertion Neighborhood

Figure 4.1: Landscapes the exchange and insertion neighborhoods of a 30 jobs instance of the JSOCMSR problem, with an objective of the global optimum of 92. In both, the neighborhood operator was applied to job 29. A closed plateau can be seen in Figure 4.1a between position 7 and 8. Open plateaus are common in Figure 4.1b, for example from position 16 to 19. Benches occur twice in Figure 4.1b at positions 9 and 26.

## 4.2 Distance Metrics

In the context of heuristic optimization, distance metrics are a common way to describe spatial characteristics of the solution landscapes. While these metrics are often used in various techniques to describe the properties of the landscape, distance metrics can also become handy throughout the search as ingredient of diversification mechanisms [HJMP00, SS05b], like in the skewed variable neighborhood search [BMU15].

More formally, the distance between two arbitrary solutions in a given landscape, is usually defined as the *minimum* number of steps required by a neighborhood operator to transform one solution into the other. Since distance metrics thus inherently depend on the solution representation and the applied neighborhood structure, obtaining exact metrics, particularly for some complex neighborhood structures is often difficult or sometimes even computationally infeasible. To this end, a common approach that is intensively applied in the literature is to fall back to surrogate metrics that provide approximations at reasonable cost.

Throughout the years, an almost uncountable number of metrics were introduced for discrete and continuous spaces, with applications in information theory [Ham50], statistics [Ken38], biology [WSB76] and many more. For neighborhood structures widely used in permutation representations, in our case primarily *exchange* and *insertion* of elements in the permutation, it is not required to deal with approximations, since exact and efficient metrics have been proposed in the literature. For the exchange neighborhood, the distance metric is based on early findings in permutation theory by *Cayley* [Cay49] regarding the number of transpositions needed to generate any permutation by applying

transpositions to the identity permutation [SS07]. For this reason, we refer to this metric in the following as *Cayley metric* [DH98]. The *Ulam* metric is the respective exact metric for the insertion move and is based on the length of the *longest increasing subsequence* in a permutation [SS07]. Both metrics are based on the group of permutations, with function composition $s \circ s'$ as binary operator [SS07] and $s^{-1}$ denoting the inverse element of $s$ in the group. In the following, both metrics will be defined briefly. For a more detailed definition, we refer to [SS07].

---

**Definition 4.2 (Cayley Metric):**
*For two permutations $s, s'$ of length $n$, the* Cayley *metric $d_{\mathcal{C}}$ is defined as*

$$d_{\mathcal{C}}(s, s') = n - c(s^{-1} \circ s') \tag{4.1}$$

*where $c(s)$ denotes the number of cycles in the permutation.*

---

---

**Definition 4.3 (Ulam Metric):**
*For two permutations $s, s'$ of length $n$, the* Ulam *metric $d_{\mathcal{U}}$ is defined as*

$$d_{\mathcal{U}}(s, s') = n - |lis(s^{-1} \circ s')| \tag{4.2}$$

*where $|lis(s)|$ denotes the length of the longest increasing subsequence of $s$.*

---

## 4.3 Landscape Characteristics and Measures

Since different phases of a search procedure are typically designed to complement each other and thus focus on different aspects, like diving deep into a local optimum or shifting the search towards unexplored regions, also the techniques commonly encountered in landscape analysis try to focus on one particular aspect rather than drawing a picture of the entire landscape at once. Therefore, in the literature some approaches have been presented that focus more on local features and may be used to understand the behavior of intensification phases, while others capture global properties, like distributions and correlations. In the following, some standard techniques and landscape characteristics will be described briefly.

**Local Perspective** The local perspective is most important for intensification mechanisms that traverse the neighborhood graph in direction of improving solution quality. As these procedures operate on a very limited subspace, they are highly affected by small structural features that trap the search in low-quality regions. The most prevalent feature of this kind are certainly local optima. Depending on the induced landscape, local optima may have different depth, basins of attraction of different sizes and may

highly vary with respect to the solution quality. Measures, like the average depth or the average solution quality of local optima may already provide useful insight into the local structures of the landscape and may allow to estimate the required effort to escape from poor local optima.

**Structural Perspective** As neighborhood search procedures often encounter several local optima on the trajectory through the landscape, interlacement and transitions between local structures affect the search on the long term. In general, a landscape is considered as rugged, if it contains a high number of intertwined local optima and shows low correlation between the objective values of neighboring solutions [Wat10]. Often, rugged landscapes tend to have rather shallow local optima compared to smooth, crater-like landscapes [Tal09].

In the literature, it is often claimed that the hardness of a problem is highly correlated to the ruggedness of a landscape [AZ00, Wat10]. Intuitively, this correlation is due the increased number of local optima that have to be explored before a global optimum may be encountered. To quantify the ruggedness, commonly approaches based on random walks are used. Techniques may perform random walks of sufficient length in the landscape and determine how often neighboring solutions end up in the same local optimum, count the number of distinct local optima in the walk or analyze distances among the obtained optima. A more advanced approach to estimate the ruggedness of the landscape is introduced in [Wei90]. Based on a random-walk through the landscape, the objective values are interpreted as a time-series, on which the autocorrelation function $r(d)$ is applied in order to estimate the correlation of the objective values between solutions with a distance $d$.

---

**Definition 4.4 (Random Walk Autocorrelation [SS05a, Sta96]):**
*For a time series $f(x_t)$ of size $m$, with mean $\overline{f}$ and variance $\sigma^2$, the autocorrelation $r(d)$ is defined as*

$$r(d) = \frac{\sum_{t=1}^{m-s}(f(x_t) - \overline{f})(f(x_{t+s}) - \overline{f})}{\sigma^2(m-s)} \tag{4.3}$$

---

The correlation length $l(d) = -\frac{1}{\ln(|r(d)|)}$ estimates the maximal distance in the walk, for which a significant correlation exists [Wat10]. Note that $l(1)$ indicates the correlation length between adjacent solutions in the search landscape. A lower correlation length close to $l(1) = 0$ indicates a rugged landscape, while a correlation length close to $l(1) = 1$ corresponds to a smooth, crater-like structure [Tal09].

**Global Perspective** From a global perspective, the overall distribution of these structures may also provide valuable insight. For some problems, it is assumed that landscapes tend to be *globally convex* [BKM94], meaning that a majority of high-quality solutions is concentrated in a rather small area of the search space. In other problems, the landscape may be rather flat and superior solutions may be evenly distributed in the search space. An approach to estimate whether the landscapes contain a massive central is to obtain a set of local optima from a random population and to compare the average distances of both populations. In landscapes, where local optima are uniformly distributed in the search space, the deviation of the average distances will not be significant with respect to the diameter of the search space. However, in *globally convex* problems, a majority of local optima will tend towards the massive central [Tal09].

### 4.3.1 Limitations

As the previous sections have pointed out, landscape analysis may become a powerful tool to extract valuable information from the problem's structure. This information can then be used to improve crucial components of the search, like neighborhood structures or diversification mechanisms. However, landscape analysis is a non-trivial task. Due to the exponential size of the solution space, considering it in its entirety usually becomes infeasible already for rather small instances, so trade-offs have to be made and thus usually one sticks to instances of moderate size, approximations and relies on a careful analysis of samples with appropriate statistical tools [Tal09].

CHAPTER 5

# Variable Neighborhood Search for the JSOCMSR Problem

This chapter describes different aspects of the devised VNS for the JSOCMSR problem. Section 5.1 starts with the solution representation and a decoding scheme to transform the concise representation into actual starting times, followed by Section 5.2, where different solution evaluation approaches are considered.

Then, Section 5.3 discusses different neighborhood structures based on standard move operators, how they can be evaluated during a systematic traversal of the solution space and compares them with respect to the computational complexity.

Based on those results, Section 5.4 then presents approaches to reduce the size of neighborhoods, enabling the search to concentrate on more promising regions during intensification. Sections 5.5 and 5.6 will then discuss initial solutions and shaking mechanisms, followed by Section 5.7 describing how all those ingredients are put together in a tailored VNS framework.

## 5.1 Solution Representation

In order to obtain an efficient neighborhood search, much has to be invested into an appropriate solution representation that is concise, but still captures all relevant aspects of a solution. Since for the JSOCMSR exact starting times are necessary to determine the makespan, one way to describe a solution could be a set of decision variables representing those starting times. Although such an approach may be natural in some MILP or CP models, for neighborhood search approaches this would induce several disadvantages concerning the domain size of the decision variables or the selection of appropriate neighborhood structures.

For the JSOCMSR a linear permutation representation is used, which is quite natural for scheduling and sequencing problems and thus often encountered in the literature. In this representation, a solution is not immediately represented by the decision variables, but is rather encoded by a linear permutation $\pi = [\pi_j]_{j=1,\dots,n}$ of the jobs in $J$, representing the total order of jobs as they acquire the common resource 0 [HRB17, HRB19]. For example, the solution shown in Figure 5.1 is encoded by the permutation $2-5-8-0-9-7-1-11-6-10-3-4$. However, to ensure that resource constraints with respect to the secondary resources are met and more importantly to derive the actual starting times of jobs and thus the makespan, a decoding scheme to convert the permutation into an actual solution must be employed.

As can be seen in Algorithm 5.1, this scheme is indeed not very complex. Starting with the first job, the procedure iterates through the permutation and schedules the jobs at the earliest possible time where constraints on the resources are satisfied. Although this approach may seem to be quite efficient, the incremental nature of how a solution is constructed turns out to be rather challenging when trying to obtain constant time neighborhood evaluation schemes.

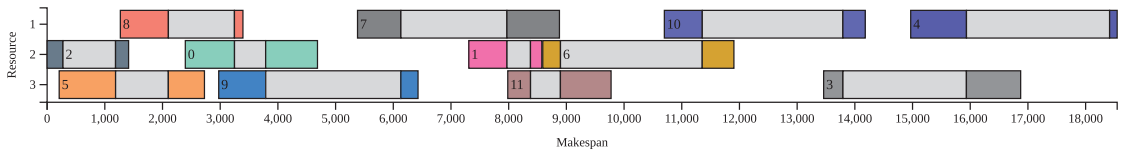



Figure 5.1: Solution of a JSOCMSR instance with $n = 12$ jobs that can be uniquely described by the permutation $2-5-8-0-9-7-1-11-6-10-3-4$.

## 5.2 Evaluation

Intuitively, performance in terms of solution quality of neighborhood search techniques relies to a significant amount on the number of conducted iterations. While this is certainly not necessarily the case for all kinds of problems and arbitrary large and complex neighborhood structures, intuitively, chances to find better solutions increase with number of visited solutions.

However, this does certainly not imply that it is sufficient to visit a large number of solutions. It is even more important to explore the search space in a systematic way, which implies that appropriate and sufficiently large neighborhoods are evaluated, enabling the search to make reasonable choices which path to follow in the search space.

Of course, this implies that the search heavily relies on the efficiency of solution evaluation, particularly to evaluate neighbors of a given incumbent solution. Although for many problems an incremental evaluation scheme allowing neighbors to be evaluated in $\mathcal{O}(1)$ time exist, the encoded solution representation and the interleaved structure of jobs made such a scheme rather challenging for the JSOCMSR problem. To this end, as an alternative to the naive solution evaluation (compare Algorithm 5.1) two further

evaluation schemes, to which we subsequently refer as (i) range-based re-evaluation (RB-EVAL) and (ii) best-effort incremental evaluation (BEI-EVAL), were studied, aiming to practically increase the efficiency of neighbor evaluation. Both approaches share some basic principles and rely on a concept to which we refer to as *synchronization* which allows to determine a position in a solutions' encoded representation, where the structural change consists only of an offset aligned among the remaining jobs in the permutation. Finally, it is important to note that one of these schemes has already been implemented by Horn et al. [HRB19] in the local optimization procedure of their A* algorithm.

**Synchronization** When comparing two solutions $s, s'$ with respect to their makespan, where $s'$ is obtained by $s$ by a change in the order as the jobs acquire the common resource, e.g. by a move operation, the difference of the makespans $\mathrm{MS}(s)$ and $\mathrm{MS}(s')$ is not apparent. However, at some point in the schedule after this structural change, the relative offsets among jobs on different resources seem to align in practice rather fast. This is due to the underlying dependency structure induced by the decoding scheme, where a job usually waits only for one, or sometimes even two preceding jobs to finish, until its resource requirements are satisfied. Once this point of *synchronization* of jobs on all secondary resources is reached, the dependency structure of subsequent jobs does not change anymore and the relative distance to the job finishing last remains. This in turn allows to directly derive the makespan in an *incremental-like* manner, as soon as this point of synchronization is reached. More precisely, this point of synchronization is the index of the last job in a *synchronization border*, consisting of a minimal set of jobs on distinct secondary resources that have aligned with respect to their position in the original solution $s$. Definition 5.1 defines the *synchronization border* more formally.

---

**Definition 5.1 (Synchronization Border):**
*Given two solutions $s$, $s'$, where $s'$ is an immediate neighbor of $s$ in some neighborhood structure $\mathcal{N}$. Assume further that the underlying permutation of jobs has only changed up to position $i$, s.t. $0 \leq i < n$. The synchronization border $\mathcal{B} \subseteq J$ is then the first set of jobs in permutation order after position $i$, satisfying the following conditions:*

1. *the set consists of at most one job per secondary resource, so $|\mathcal{B}| \leq m$ and $\forall j, j' \in \mathcal{B} : q_j \neq q_{j'}$*
2. *if a secondary resource $r$ runs out of jobs before the last job in the permutation has been reached, the synchronization border only consists of jobs in $J'$, s.t. $J' = \{j \mid q_j \in R \setminus r\}$*
3. *the jobs are aligned with respect to their individual offsets in $s$ and $s'$, i.e. $\exists c \in \mathbb{Z} \;\; \forall j \in \mathcal{B} : s_j - s'_j = c$*

---

Figure 5.2 illustrates such a scenario, where in a schedule of 12 jobs, job 4 is moved forwards between 0 and 9. Depending on the situation, the inserted job then has to be

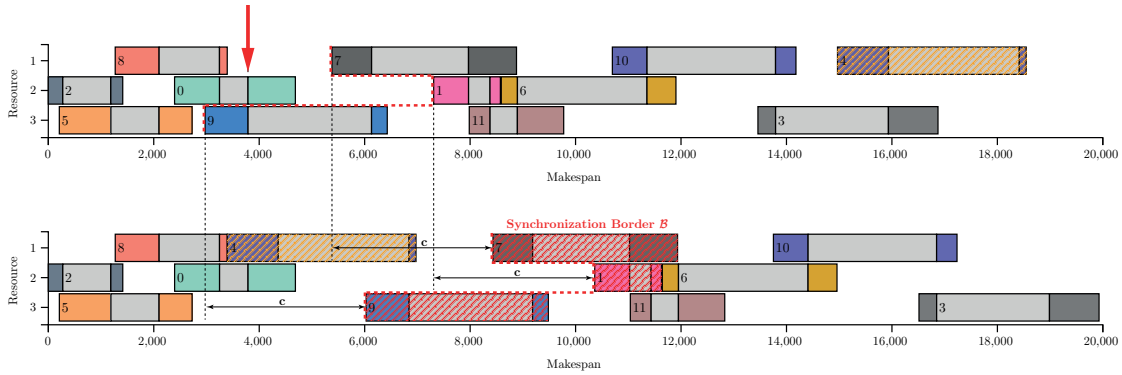



Figure 5.2: Example of a synchronization border in a solution with $n = 12$ jobs. The red arrow indicates the destination position of job 4 in a neighboring solution. The synchronization border consists of jobs 7, 1 and 9, which is the first set of jobs on different secondary resources, where the offsets with respect to their position in the incumbent solution have aligned to some $c$. After this synchronization border, no unaligned job offsets are encountered, expect for job 4.

scheduled to meet the resource constraints and thus may in turn delay latter jobs due to conflicts on resources 0, $r \in R$ or transitive propagations. In the example, the insertion of job 4 delays job 9 and subsequent jobs. However, the decoding scheme ensures that the schedule is as compact as possible, which in turn induces heavy dependencies among subsequent jobs, i.e. a job waits only for its immediate predecessor on resources 0, $r \in R$ or coincidentally on both to finish, like job 6.

So even though structural changes are introduced into the schedule at some positions, the dependencies among subsequent jobs usually align to their previous interleaving pattern rather soon. If this alignment is established among all secondary resources, the relative positions of subsequent jobs do not change anymore. In the example, the *synchronization border* is indicated by the dashed red line. It is, however, important to note that this assumes that the job(s) inducing the structural change, i.e. job 4 in the example, are not part of the schedule anymore. Based on the evaluation scheme, as will be described shortly, this is achieved either by re-evaluating entire sections of a solution subject to structural change or by making use of partial solutions not including the respective jobs.

**Evaluation Schemes** Based on the *synchronization* mechanism, two evaluation schemes, namely RB-EVAL and BEI-EVAL were devised and implemented for the neighborhood structures presented later on. On the one hand, RB-EVAL evaluates the entire subsection of a solution that is subject to structural changes followed by a single synchronization step. An illustration is given in Figure 5.3. This reduces preparatory tasks, as they are required in BEI-EVAL, to a minimum, although the scheme degenerates into a naive evaluation for neighbors constructed by structural changes of large spatial scope, e.g. moving the last job to position 0 in the permutation. On the other hand, BEI-EVAL avoids this by

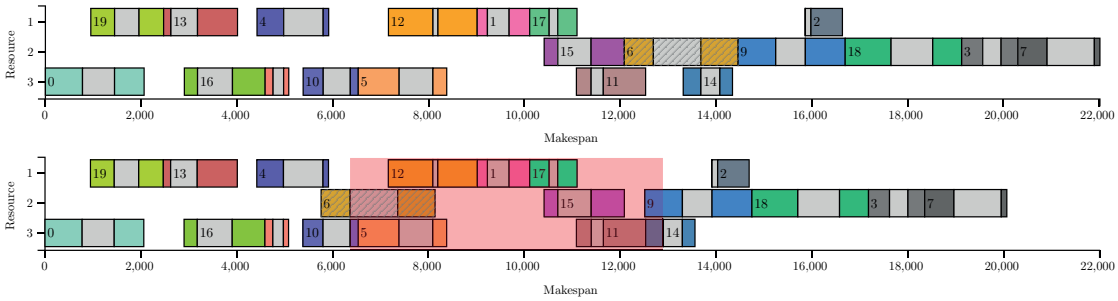



Figure 5.3: Example of a range-based re-evaluation in a solution with $n = 20$ and $m = 3$. Job 6 is moved from position 13 to 7, inducing a significant makespan reduction as well as some structural changes in the job dependencies. The makespan of the neighboring solution is determined by re-evaluating the range from position 7 to 13 followed by a synchronization step. Here, the synchronization border consists only of jobs $\mathcal{B} = \{2, 9\}$.

making use of partial solutions not including the job subject to the move operation or conducts multiple synchronization steps, making it more appropriate particularly in large instances.

In addition to *synchronization*, both approaches heavily rely on auxiliary data structures that allow to efficiently lookup the earliest starting time for a job at a given position in the permutation as well as to determine the relative distance of a job in the permutation to the makespan. Depending on the concrete neighborhood structure, these data structures have to be prepared once or even $n$ times per incumbent solution. Even though this induces significant computational cost, for the studied instances this still led to a major improvement in the neighborhood evaluation's efficiency.

It is, however, important to note that in case of significant resource imbalance (e.g. one secondary resource containing only a single job scheduled last), the evaluation schemes will degenerate into a naive solution evaluation with an additional overhead for preparatory tasks and the identification of the synchronization border.

**Algorithm 5.1:** Permutation decoding procedure to derive exact starting times given a permutation $\pi$.

**Input:** An instance of the JSOCMSR, a permutation $\pi = [\pi_j]_{j=1,\dots,n}$ of jobs
**Output:** A solution $s[s_j]_{j \in J}$ described by exact starting times for the jobs in permutation $\pi$.

```
1  est = [est_i = 0]_{i=0,...,m}
2  s = [s_i = 0]_{i=1,...,n}
3  for i ← 1 to n do
4  |   j ← π[i]
5  |   s[j] ← est[0] − p_j^pre
6  |   if est[q_j] > s[j] then
7  |   |   s[j] ← est[q_j]
8  |   end
9  |   est[0] ← s[j] + p_j^pre + p_j^0
10 |   est[q_j] ← s[j] + p_j
11 end
```

## 5.3 Neighborhoods

When choosing neighborhood structures for optimization problems, there are obviously plenty of possibilities. While highly specific neighborhoods may indeed be advantageous since they exploit particular properties of the problem, often standard moves appropriate for representation already suffice. The simplicity of those moves may even become advantageous with respect to solution evaluation schemes, e.g. for incremental evaluation. For scheduling problems based on linear permutation representations the standard neighborhoods most widely used are among others based on *inserting* and *exchanging* of jobs [dBS01].

### 5.3.1 Insertion

The insertion neighborhood consists of any permutation obtained by moving a single job to different position and thus has a cardinality of $\mathcal{O}(n^2)$. As can be seen in Figure 5.4, for the JSOCMSR problem the neighborhood may allow to fill gaps in the schedule that have been induced due to contention on secondary resources, like between jobs 4 and 8.

Considering moves in any direction for any job, however, turned out to be relatively expensive in terms of evaluation. One way to handle this is by restricting moves to one direction, i.e. only *forwards* or *backwards*. This way, the evaluation scheme can be improved significantly, requiring only a single synchronization step with a brief preparation of auxiliary data structures once for each job per neighborhood evaluation. The neighborhood for arbitrary moves can still be constructed as a combination of the restricted ones, but is obviously more expensive in terms of computation time and as will

be seen later on in Chapter 7, is particularly within VND schemes not worth the additional cost. A formal definition of the neighborhood structure is given in Definition 5.2.

Finally, in order to be able to efficiently evaluate entire insertion neighborhoods completely, the respective neighbors have to be generated in an incremental manner by swapping subsequent jobs in the permutation to avoid excessive move operations in the underlying array data structure containing the permutation.

---

**Definition 5.2 (Insertion [SS07]):**
*For a given permutation* $\pi = [\pi_j]_{j=1\ldots n} \in \mathcal{S}$ *the insertion move of* $\pi_i$ *to position* $j$, $i \neq j$ *is defined by a function* $\delta_I^{i,j} : \mathcal{S} \to \mathcal{S}$, *s.t.*

$$\delta_I^{i,j} = \begin{cases} (\pi_1 \ldots \pi_{i-1}\ \pi_{i+1} \ldots \pi_j\ \pi_i\ \pi_{j+1} \ldots \pi_n) & \text{if } i < j \\ (\pi_1 \ldots \pi_j\ \pi_i\ \pi_{j+1} \ldots \pi_{i-1}\ \pi_{i+1} \ldots \pi_n) & \text{if } i > j \end{cases}$$

*In case* $i > j$, *we refer to the move as* forward-insertion *and for* $i < j$ *as* backward-insertion.

*To the neighborhood induced by the move* $\delta_I^{i,j}$, *we refer as* $\mathcal{N}_I$, $\mathcal{N}_F$ *and* $\mathcal{N}_B$ *for arbitrary, forward and backward insertions respectively.*

---

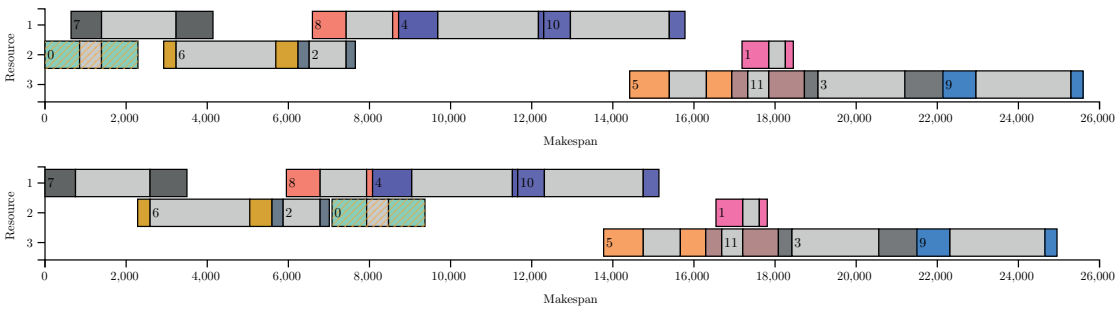



Figure 5.4: Example of an insertion move, where job 0 is moved backwards to fill a gap on the common resource due to contention on secondary resources between jobs 4 and 8.

### 5.3.2 Swap and Exchange

The swap neighborhood structure, often also referred to as *transpose*, consists of all $n-1$ permutations generated by swapping adjacent elements in the given permutation. Due to its limited size and structural change in the permutation, the impact of a swap move may not always be of most significance with respect to the change in the objective value. The limited size, although, may still allow it to be feasible even for large instances. For the JSOCMSR, the swap neighborhood may be advantageous, since evaluation of the neighbor solutions is generally rather expensive.

As a generalization of *swap*, the *exchange* or *interchange* neighborhood structure is defined as the set of permutations obtained by exchanging any pair in the given permutation.

Thus, the size of the neighborhood is $n(n-1)/2$. An exact definition is given in Definition 5.3. Figure 5.5 shows an example schedule with one improving neighbor interchanging jobs 9 and 10.

For the exchange neighborhood the BEI-EVAL evaluation schema is implemented with a dual-synchronization approach, allowing evaluation for the considered instances practically in constant time, as will be shown in Chapter 6. However, in the worst case, evaluation still costs $\mathcal{O}(nm)$. As an alternative, the RB-EVAL evaluation scheme turns out as practically relevant for moves where the distance $\delta = j - i$ between jobs $\pi_i$ and $\pi_j$ is moderate, primarily since only a single synchronization border has to be identified.

---

**Definition 5.3 (Exchange [SS07]):**
*For a given permutation $\pi = [\pi_j]_{j=1\ldots n} \in \mathcal{S}$ the exchange move of jobs $\pi_i$ and $\pi_j$, $i \neq j$, is defined by a function $\delta_X^{i,j} : \mathcal{S} \to \mathcal{S}$, s.t.*

$\delta_X^{i,j} = (\pi_1 \ldots \pi_{i-1}\ \pi_j\ \pi_{i+1} \ldots \pi_{j-1}\ \pi_i\ \pi_{j+1} \ldots \pi_n)$
*In case $|i-j| = 1$, we refer to the move as a* swap.

*To the neighborhood induced by the move $\delta_X^{i,j}$, we refer as $\mathcal{N}_X$ and $\mathcal{N}_S$ for exchanges and swaps respectively.*

---

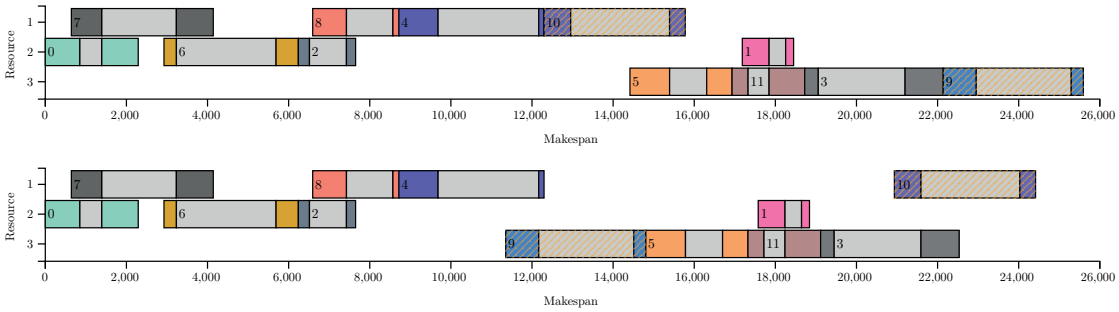



Figure 5.5: Example of an exchange move, where jobs 9 and 10 change position in the permutation $\pi$ defining the solution.

### 5.3.3 Inversion

Finally, the last standard neighborhood considered in this work is the inversion of a subsequence of length $k$. The incremental-like evaluation of a neighbor requires the iteration of the entire reversed subsequence of $k$ jobs, followed by a single synchronization. As can be seen for example in Figure 5.6, a single move introduces significant structural changes into a solution. Even though this leads to a rather rugged search landscape, as will be shown in Chapter 7, this behavior may still be advantageous to diversify the search in VND or during the shaking phases.

**Definition 5.4 (Inversion):**
*For a given permutation* $\pi = [\pi_j]_{j=1...n} \in \mathcal{S}$ *the subsequence inversion move of given by* $i$ *and* $k$, $i \leq n - k$ *is defined by a function* $\delta^{i,k}_{INV} : \mathcal{S} \to \mathcal{S}$, *s.t.*

$\delta^{i,k}_{INV} = (\pi_1 \ldots \pi_{i-1}\ \pi_{i+k}\ \pi_{i+k-1} \ldots \pi_{i+1}\ \pi_i\ \pi_{i+k+1} \ldots \pi_n)$

*To the neighborhood induced by the move* $\delta^{i,k}_{INV}$, *we refer as* $\mathcal{N}_{INV-k}$.

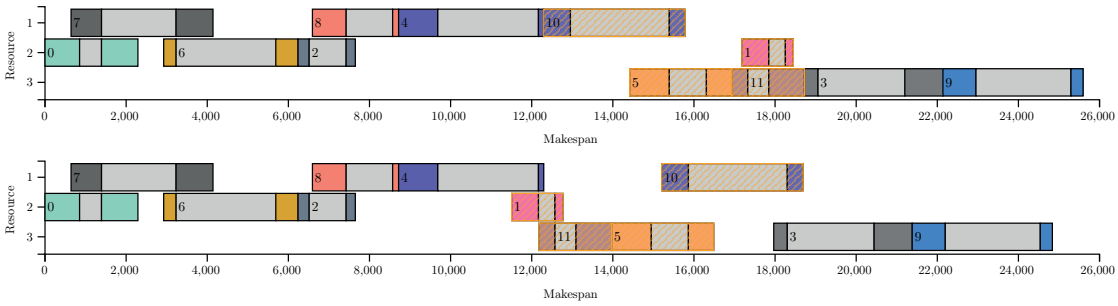



Figure 5.6: Example of an inversion move, where a subsequence of 4 jobs is inverted, coincidentally leading to an improved interleaving at the boundaries of the subsequence.

## 5.4 Reduction of Neighborhoods

The previous section showed that for the JSOCMSR, evaluating neighborhoods may become rather expensive depending on the structure and the size of the instances. However, in practice it is both important to explore large neighborhoods to make a step towards rather steepest descent, while also evaluate the neighborhoods fast to explore large areas of the search space. When the instance size increases, it becomes apparently even harder to satisfy both of those criteria and soon this affects the performance of the VNS in a negative way.

Stepping back to smaller, more tractable neighborhoods is also not really a viable option. Fortunately, for the JSOCMSR it turns out that some moves inherently do not contribute to the improvement of a solution. So it may be a promising approach to try to determine those candidate solutions and exclude them from being evaluated in the first place. This way, one can find a good compromise between the desire to evaluate relatively large neighborhoods, while also being able to evaluate them quite efficiently. In the following, different approaches to prune neighborhoods will we presented. However, not all of them showed to be of practical relevance and were thus not considered in the computational study in Chapter 8.

### 5.4.1 Heuristic Neighborhood Pruning

The first rather simple approach tries to reduce the size of the neighborhoods by applying certain heuristics based on properties of a solution.

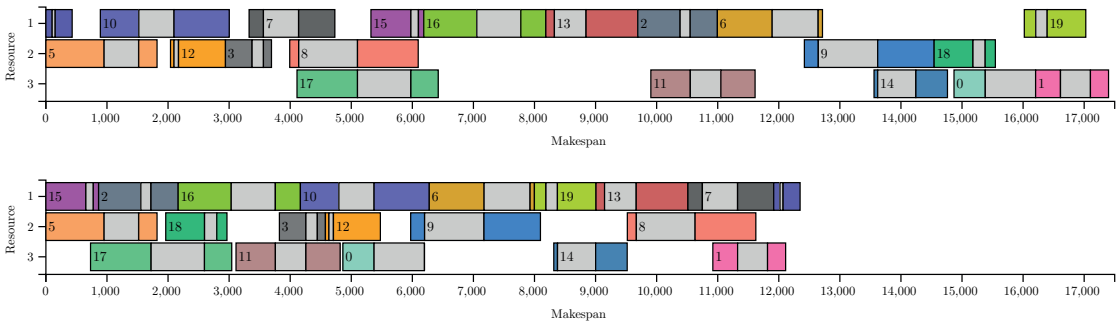



Figure 5.7: Example of an arbitrary solution and an optimum for an instance with 20 jobs and three secondary resources. The example illustrates the characteristic that superior solutions produce more compact schedules by interleaving jobs on different secondary resources to avoid gaps on the common resource 0.

**Forced Interleaving** One reason why some resources, both the common resource 0 or the secondary resources $r \in R$, remain unutilized is due to delays induced by consecutive jobs on the same secondary resource. In case the jobs are non-trivial in size compared to others, this inherently produces phases where some of the resources and at least resource 0 remain idle, which in turn may have a negative effect on the makespan.

Such a scenario is illustrated in Figure 5.7. While some sections in the first solution seem to be rather compact and utilize resource 0 optimally, the congestion on resource 1 due to the sequence of jobs 15, 16, 13 and 2 produces a long phase, where resources 2 and 3 remain completely idle, causing in turn significant gaps on the common resource. It is apparent that such a sequence is highly unlikely to constitute optimal solutions for most instances. The corresponding optimal solution, on the other hand, shows to be rather compact due to the tendency that jobs of different secondary resources are scheduled alternatingly, allowing for immediate transitions on resource 0 and little to no congestions on the secondary resources.

Based on this observation, several heuristics were devised that foster moves to force an interleaving of jobs of different secondary resources onto a solution. The approaches either exclude moves based on the immediate predecessors or successors of a job's current or respective destination position. However, it is important to note that such a forced interleaving of jobs on distinct secondary resources is still only of heuristic nature and not necessarily the case in optima of any instance. For example, in instances where a large number of short jobs are to be scheduled on some secondary resource $r$ and significantly fewer jobs on $r'$ with short utilization of the common resource 0, then the heuristic will inherently force a suboptimal interleaving of jobs. To this end, the heuristic may only

be employed in VND schemes, where subsequent neighborhoods can produce improving moves, once the heuristic guides the search into poor local optima.

**Gaps on resource** 0 For a given solution, a naive lower bound for the makespan is $\sum_{j=1}^{n} p_j^0$, assuming that the common resource 0 has an optimal utilization such that for all consecutive jobs $j, j', j'$ immediately acquires resource 0, as soon has $j$ has released it [HRB17].

Although this is indeed desirable, due to the additional constraints introduced by the secondary resources, eliminating resource gaps on the common resource is apparently not always possible, as illustrated in Figure 5.8.

However, when minimizing the makespan, removing utilization gaps on the common resource may serve as an appropriate heuristic. So another way to reduce the size of the neighborhood is to only consider moves, where jobs cause utilization gaps on the common resource, since their removal could potentially induce an improvement of the makespan. Alternatively, the positions of gaps in a schedule may also be used to determine potential destination positions of jobs in diversification phases of the search.

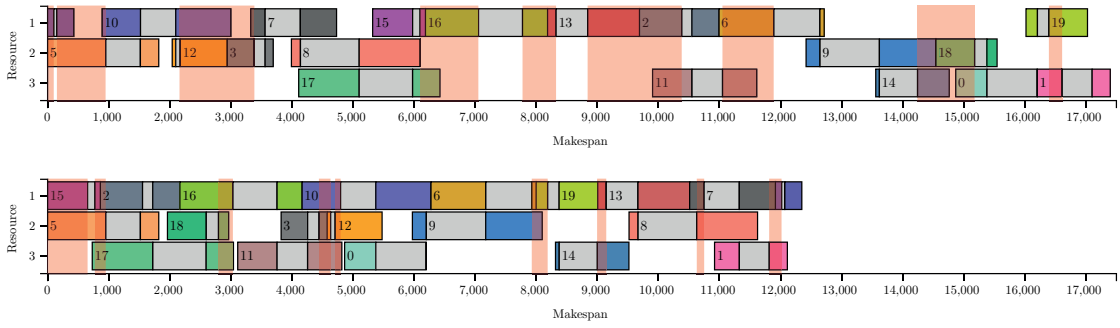



Figure 5.8: Example of an arbitrary solution and an optimum for an instance with 20 jobs and three secondary resources. The example illustrates that utilization gaps on the common resource are minimized in high quality solutions.

### 5.4.2 Reduction of Move Distances

An alternative approach to reduce the size of neighborhoods is by restricting the maximum distance of the underlying move operations. So instead of allowing, for example, exchanges between arbitrary jobs $j$ and $j'$ in a solution, the distance $\delta = |\text{pos}(j) - \text{pos}(j')|$ between $j$ and $j'$, where $\text{pos}(j)$ indicates the position of job $j$ in the permutation, is bounded by a constant $c$.

For the JSOCMSR this has several advantages. As will be shown in Chapter 6, the evaluation of a move can be performed quite fast, if only a limited subsequence of the solution has to be evaluated, in which case the overhead due to potentially multiple synchronization steps would dominate the evaluation of the subsequence.

Limiting the move distance apparently also reduces the size of the neighborhood, which may be beneficial at least in terms of the execution time required to evaluate an entire neighborhood. For example, the cardinality of an exchange neighborhood with size $n(n-1)/2$ then becomes $nc - (c^2 + c)/2$. It is obviously not easy to decide on an appropriate value for $c$ beforehand, however, such a constant is typically subject to manual tweaking or automated parameter tuning.

However, this approach also comes with a severe disadvantage. By limiting the move distance to a maximum of $c$, some potentially improving moves are not evaluated anymore, which inherently introduces new local optima in the search space. To overcome this limitation, either intensification phases have to be extended by neighborhoods based on unlimited moves, or the search heavily relies on shaking phases that probabilistically transfer the incumbent solution into the desired regions. As another way to overcome this limitation, we experimented with some approaches that try to adaptively adjust $c$ throughout the search or even the neighborhood evaluation, in order to both limit the spatial scope in favor of efficiency, while allowing it to increase in scenarios where the number of improving solutions in the trajectory drops. A more thorough investigation of this approach is left for future work.

### 5.4.3 Critical Jobs

When taking a look at Figure 5.9, it can be observed that some jobs do not have any immediate impact on the makespan of the solution. So, while changing the position of those job can indeed be disadvantageous, the dependency structure in the solution actually prevents improving moves. As an example, consider the jobs in Figure 5.9 with the gray overlay. Moving job 11 between 17 and 0 apparently extends the makespan. However, the dependency between jobs 15 and 6 actually prevents improving moves of job 11. On the other hand, jobs like 9 actively define the makespan and thus moving them, for example between 10 and 7, has a significant impact on the objective value.

As it turns out, schedules actually contain two types of jobs, the ones that define the makespan and the others that just fill some gaps, but cannot be moved in a way to improve the makespan. Horn et al. [HRB19] describe this characteristic by means of a dependency graph, where the *critical jobs* are those defining the makespan of the solution. A formal definition is provided in Definition 5.5. The corresponding dependency graph for the schedule shown in Figure 5.9 is shown in Figure 5.10.

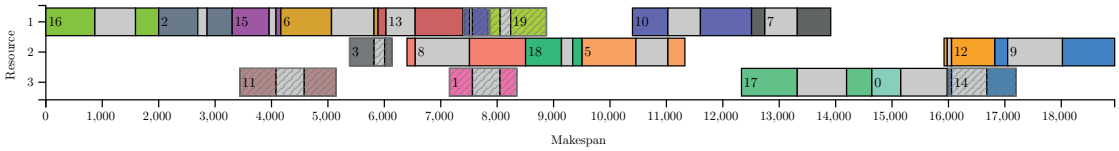



Figure 5.9: Solution of an instance with 20 jobs and three secondary resources, where jobs not in the critical set are indicated by a gray overlay. For example, job 11 could be moved between 15 and 6 almost arbitrarily without affecting the makespan at all.

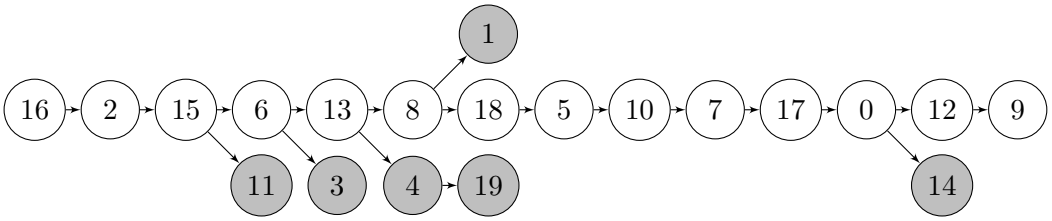



Figure 5.10: Dependency graph induced by the schedule shown in Figure 5.9. Jobs not part of the critical set are again illustrated in gray.

---

**Definition 5.5 (Dependency graph, critical paths and critical jobs):**
*For a given solution $s$, the corresponding dependency graph $\mathcal{G}_\mathcal{D} = (\mathcal{V}_\mathcal{D}, \mathcal{E}_\mathcal{D})$ consists of a set of vertices $\mathcal{V}_\mathcal{D}$ corresponding to the jobs in $s$ and a set of edges $(i, j) \in \mathcal{E}_\mathcal{D}$ representing direct dependencies among consecutive jobs, i.e. $(j, j') \in \mathcal{E}_\mathcal{D}$ iff job $j'$ utilizes either the common or the secondary resource immediately after it has been released by job $j$. In this case we say $j'$* depends *on $j$.*

*In the dependency graph, the makespan is defined by* critical paths*, which are all paths from the first job to any job finishing last on a resource.* Critical jobs *are in turn all jobs, whose corresponding vertices lie on all critical paths [HRB19]. In the following, we refer to the set of critical jobs as* critical set $\mathcal{C}$*.*

---

**Theorem 5.1:**
*For a solution $s$ with a critical set $\mathcal{C}$ and a job $j$, s.t. $j \notin \mathcal{C}$, the removal of $j$ does not reduce the makespan.*

**Proof:** Since $j$ is not in the critical set, it follows that either (i) there does not exist a job $j'$ that depends on $j$ or (ii) there exists a job $j'$ depending on $j$ that also depends on third job $j''$.

Case (i): If $j'$ does not exist and thus no job is waiting for $j$ to finish, removing $j$ from the schedule does not induce any structural change in the schedule and thus the makespan remains unchanged.

Case (ii): If $j'$ exists, but $j \notin \mathcal{C}$, it follows that $j'$ also depends on exactly one $j''$ due to the given resource structure in the JSOCMSR. So coincidentally, $j$ and $j''$ release resource 0 and $r_i, i = 1, \ldots, m$ respectively, exactly with an offset of $p^{\text{pre}}_{j'}$ and thus $j'$ depends on both of them. When removing $j$ from the schedule, $j'$ still depends on $j''$ and thus the makespan again remains unchanged.

**Theorem 5.2:**
*For a solution $s$ with a critical set $\mathcal{C}$ and a job $j$, s.t. $j \notin \mathcal{C}$, an* insertion *move cannot reduce the makespan.*

**Proof:** An *insertion* move is composed of the removal of $j$ and the reinsertion at a new position $l$. Since $j \notin \mathcal{C}$, it follows from Theorem 5.1 that the removal of $j$ does not have any impact on the makespan. So $\text{MS}(s) = \text{MS}(s_p)$, with the partial solution $s_p = s \setminus j$. For a given solution $s_p$, it trivially follows that the insertion of a new job $j$ cannot reduce the makespan.

**Theorem 5.3:**
*For a solution $s$ with a critical set $\mathcal{C}$ and jobs $j, j'$, s.t. $j, j' \notin \mathcal{C}$, an* exchange *move of $j$ and $j'$ cannot reduce the makespan.*

**Proof:** An *exchange* move is composed of the removal of both $j, j'$ and the reinsertion at the respective other positions. Since $j, j' \notin \mathcal{C}$, it follows from Theorem 5.1 that the removal of both $j, j'$ does not have any impact on the makespan. Similar to Theorem 5.2, it follows that the reinsertion of both jobs at the respective new position cannot decrease the makespan.

Based on these results, the critical set can then be used to avoid the evaluation of certain candidate solutions in the neighborhood. For *insertion* and *exchange* neighborhoods, all those moves can be skipped, where the jobs being removed as part of the move are not in the critical set. Since determining the critical set has a complexity of $\mathcal{O}(mn)$ [HRB19], for smaller instances with quite efficient evaluation schemes, reducing the neighborhood with the critical set may induce a significant overhead.

## 5.5 Initial Solution

Due to the nature of the selected neighborhood structures, our VNS acts as a pure improvement metaheuristic that starts from a feasible initial solution. In general, we consider two different approaches. Their impact on the performance of the VNS will be studied in Chapter 8 in more depth.

### 5.5.1 Randomized Solution

Probably the simplest approach to construct a feasible solution for the JSOCMSR is to generate a random permutation. Although this approach does not consider any problem specific characteristics and thus the initial objective value is expected to be far from objective values obtained from optimal solutions, the solution is not biased by any construction heuristics that could potentially force the search into an area of local optimal solutions from which it is hard to escape from.

In particular, randomized initial solutions could become advantageous in small or medium sized instances, where local optima a rather shallow, as will be shown in Chapter 7, and thus the search soon reaches shaking phases, such that the search space can be explored in a more strategic way.

In large instances, however, this advantage could soon turn into the opposite. Since the solution is completely unstructured, it is likely that the first intensification phases must

perform quite a lot of local improvements, until a local optimum is eventually reached. In case this intensification phase exploits several neighborhood structures in a VND scheme, a significant amount of the available computation time may already be consumed, before the actual strategic exploration of the search space has even started.

### 5.5.2 Least Lower Bound Heuristic

A commonly used approach to obtain initial solutions for improvement heuristics is by construction heuristics. In this work, we rely on the least lower bound heuristic (LLBH) devised by Horn et al. [HRB17, HRB19]. Their approach constructs a solution by greedily selecting one job at a time using a vector of lower bounds on the makespan objective as search guidance.

## 5.6 Shaking

Besides an efficient and yet effective intensification phase, shaking is a crucial component in VNS. By introducing some structural change into the solution, usually in a probabilistic manner, the shaking phase primarily aims to escape from local optima, once the intensification gets stuck in suboptimal regions. Furthermore, the shaking procedure has to be designed in a way to comply with the underlying structure of the search space. For example, if high quality solutions are concentrated in small areas of the search space, a shaking procedure focusing on small surroundings around the incumbent will most likely struggle to find significantly improving solutions. To this end, we dedicated an entire chapter of this work to an analysis of the search space to at least shed some light on aspects we think are important to design a proper shaking procedure. Although the results of the landscape analysis will be presented in more depth in Chapter 7, we start this section with a brief outlook on an important aspect affected the way the shaking procedure was designed.

In general, high quality solutions for the JSOCMSR are usually widely distributed in the search space. To this end, when encountering a locally optimal solution, it is not unlikely to find improving solutions already in close areas around the current optimum. Furthermore, for instances of limited size (i.e. this does not necessarily hold true for large instances) it could be shown that even globally optimal solutions are distributed across the entire search space. As a consequence, a primary goal of the shaking procedure shall be to investigate relatively close surroundings of the incumbent in the beginning and then gradually explore more and more distant areas around it, until improving solutions are eventually encountered.

### 5.6.1 Random Moves

A rather common, but yet effective shaking technique is to perform randomized transitions from the incumbent solution in a given neighborhood [HMTH17]. By increasing the number of randomized moves in each of the shaking neighborhoods, this approach tries to

systematically investigate more distant areas in the search space, until a more promising region is eventually identified.

For shaking neighborhood structures for the JSOCMSR we again rely on the three fundamental intensification neighborhood structures, adapted in a way to repeatedly apply up to $k$ randomized transitions. A definition is given in Definition 5.6.

---

**Definition 5.6 (Shaking Neighborhoods):**
*For a given neighborhood structure $\mathcal{N}_{\mathcal{M}}$, where $\mathcal{M} \in \{I, X, INV\}$, the corresponding shaking neighborhood structure $\mathcal{N}^S_{\mathcal{M},k}$ is defined as a neighborhood structure that consecutively performs $k$ randomized transitions in the neighborhood induced by $\mathcal{N}_{\mathcal{M}}$.*

---

As will be shown in Chapter 7, the analysis of the search landscapes showed that the considered neighborhood structures show quite different characteristics with respect to the destructiveness the moves introduce into a solution. For example, while a single insertion of a job to some new position may in some cases not even change the makespan of a solution, i.e. a perturbation does not have any immediate effect, inversion moves may be considered to lie on the opposite side of the spectrum, since the structure of the inverted sequence is completely destroyed with a possibly high impact on the makespan.

For the JSOCMSR, the idea is now to exploit these characteristics and combine the different shaking neighborhood structures in an interleaved manner. This allows to gradually introduce more and more structural changes into a solution in relatively a fine-grained manner as the shaking phase advances. An example of such an interleaved combination of different shaking neighborhood structures is given in Table 5.1.

| $i$ | Neighborhood Structure |
|---|---|
| 1 | $\mathcal{N}^S_{I,k=1}$ |
| 2 | $\mathcal{N}^S_{X,k=1}$ |
| 3 | $\mathcal{N}^S_{I,k=2}$ |
| 4 | $\mathcal{N}^S_{X,k=2}$ |
| ... | |
| $k_{\max} - 1$ | $\mathcal{N}^S_{I,k=n}$ |
| $k_{\max}$ | $\mathcal{N}^S_{X,k=n}$ |

Table 5.1: An example how shaking neighborhoods with different destructiveness are combined in an interleaved manner, to continuously increase the structural change with increasing $i$.

Although neighborhood structures and interleaving patterns are an important aspect, the number of shaking neighborhoods and the distribution of $k$ play an at least as important role in our shaking procedure. On the one hand, fast degeneration into random-restart should in general be avoided, although at some point it still should theoretically be

possible to reach any solution in the search space. To this end, we apply a scheme that allows to fixate the total number of shaking neighborhoods $k_{\max}$ and distributes $k_{i=1\ldots k_{\max}}$ in an exponential way.

$$k_i = \left\lceil e^{\frac{i \cdot log(n)}{k_{\max}-1}} \right\rceil \tag{5.1}$$

Even more, the impact of parameter $k_{\max}$ must not be neglected. On the one hand, for small and medium sized instances, where the primary goal is to some extent to find the global optimum, a large number of shaking neighborhoods can be employed to thoroughly investigate the search space. Due to the efficiency of the intensification phase, the relatively high number of shaking neighborhoods and even large perturbations are still feasible. For large instances, however, incorporating a high number of shaking neighborhoods may turn out to be disadvantageous, due to the increased computational cost induced by subsequent intensification phases.

Furthermore, performing intensive perturbations with $k$ close to $n$ is intuitively not very useful, since essentially a random restart is conducted, causing probably once more a relatively long initial intensification phase until a — possibly poor — local optimum is obtained, from which slow shaking procedures then continuously have to refine the solution again. The number of shaking neighborhoods as well as the number of perturbations performed in each will be presented in Chapter 8.

### 5.6.2 Intensified Shaking

Although Chapter 8 shows that performing purely randomized moves in a continuously more perturbative manner performs quite well for a wide range of instances, performing shaking moves in a more strategic or *intensified* [HMTH17] way, may become advantageous with respect to various aspects.

On the one hand, focusing shaking rather on promising moves may increase the success rate of the iterations, i.e. shakes where subsequent intensifications could improve the objective, which in turn could have a positive impact on the runtime.

Even more, by performing shakes in a more systematic, but still probabilistic way, we aim to find a way out of rather *strong* local optima, where purely randomized moves could not obtain any improvement in reasonable time. In particular, the ideas for these extensions originate from experiments, where for some particularly hard instances our GVNS did not manage to come even close to the globally optimal solutions obtained by the A* algorithm of Horn et al. [HRB19]. To this end, several approaches have been devised that try to perform shaking in more promising way, while still preserving the probabilistic nature.

**Intensified Random Moves** Shaking neighborhoods based on random moves were adapted to select jobs with a probability increasing with the utilization gap the job induces on resource 0. For jobs immediately acquiring the common resource after their previous job, a gap of 1 is assumed. Thus, in cases a solution is subject to large utilization gaps on the common resource, the probability of jobs that acquiring the common resource remains rather low, while those jobs become more relevant as a solution is improved and thus the utilization of the common resource increases.

Algorithm 5.2 shows the procedure to derive a vector of probabilities already in an aggregated integer form to avoid floating point operations. Figure 5.11 illustrates an example of this approach, where in a solution with 20 jobs utilization gaps are highlighted red, while immediate transitions are indicated by a green peak.

**Algorithm 5.2:** Procedure to prepare a vector of probabilities for intensified shaking.

**Input:** A solution $s$
**Output:** A vector $p$ containing probabilities already in prefix-aggregated form.

$gaps = [gaps_i = 0]_{i=1,\dots,n}$;
$gaps \leftarrow \text{DeriveGaps}(s)$ ;
$p = [p_i = 0]_{i=1,\dots,n}$;
$previous \leftarrow 0$;
**for** $i \leftarrow 1$ **to** $n$ **do**
  **if** $gaps[i] > 0$ **then**
    $p[i] \leftarrow previous + gaps[i]$;
  **else**
    $p[i] \leftarrow previous + 1$;
  **end**
  $previous \leftarrow p[i]$;
**end**
**return** $p$;

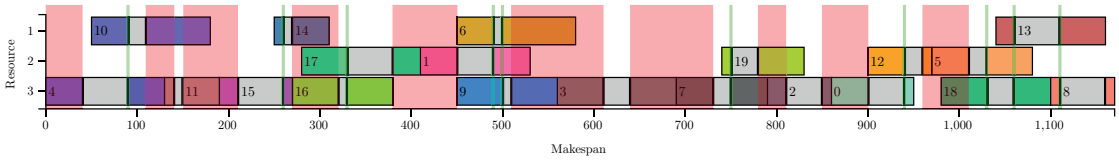



Figure 5.11: Solution of an instance with $n = 20$, where utilization gaps on resource 0 are highlighted in red, while immediate transitions are illustrated by a green peak.

Based on this approach to select jobs in suboptimal regions of a solutions, the following shaking neighborhoods were devised:

1. $\mathcal{N}^S_{\text{G}-I}$: insertion move incorporating the adapted probability function

2. $\mathcal{N}^S_{\text{G}-X}$: exchange move incorporating the adapted probability function
3. $\mathcal{N}^S_{\text{G}-INV-k}$: aggregates utilization gaps of $k$ consecutive jobs to derive the probability of a subsequence being selected.

**Destroy and Repair** Once trapped in particularly strong local optima, e.g. due to the LLBH construction heuristics, for the JSOCMSR it turned out to be quite hard to explore improving areas by conducting random moves only. To this end, we experimented with two approaches based on destroy and repair principles, as in iterated greedy (IG) algorithms [RS07], to restructure solution subcomponents in order to transfer the search into high potential, but yet relatively new areas of the search space.

In our work, shaking based on destroy and repair like principles is in its essence based on three phases: (i) the extraction of a proper subproblem, (ii) obtaining an improved solution for this subproblem and finally (iii) reinserting the solution of the subproblem. A sketch of the basic steps are shown in Algorithm 5.3.

**Algorithm 5.3:** Intensified shaking by a destroy and repair like mechanism to explore entirely new areas in the search space.

**Input:** A solution $s$ described by a permutation $\pi = [\pi_j]_{j=1\ldots n}$ of jobs, the cardinality of the shaking neighborhood $k$
**Output:** A solution $s'$ as a result of the shaking procedure.

```
1 s_p ← Extract(s, k)
2 s_p^* ← Optimize(s_p)
3 s' ← Merge(s \ s_p, s_p^*)
4 return s';
```

A partial solution is selected based on a probability increasing with the time the common resource remains unutilized, similarly to Algorithm 5.2. To this end, for each of the $n - k + 1$ consecutive partial solutions $s_p$ in $s$, we derive a probability corresponding to the time resource 0 remains idle. Based on this probability distribution, a subproblem is then selected at random.

Subsequently, a good or even optimal solution of the subproblem is obtained, either with IBM ILOG CPOptimizer or with the LLBH. Finally, the subsequence is reinserted at its original position. For the CP model the formulation presented by [HRB19] was used, in addition to constraints enforcing the offset due to jobs preceding the subsequence.

An example, of a local optimum obtained with a local search based on $\mathcal{N}_{\mathcal{B}}$, where a subsequent intensification phase after the repair and destroy based shaking procedure could further improve the objective is shown in Figure 5.12. The overlay highlights the consecutive section of jobs that was reinserted into the solution in a locally optimized manner. Although the makespan of the overall solution has increased due to the reinsertion, the operation has changed the structure of the solution in a way, such that subsequent intensification phases can proceed. In the following, we refer to the shaking

neighborhoods following this procedure as $\mathcal{N}^S_{\text{CP},k}$, in case IBM ILOG CPOptimizer is used to obtain a solution for $s_p$ and $\mathcal{N}^S_{\text{LLBH},k}$ for LLBH.

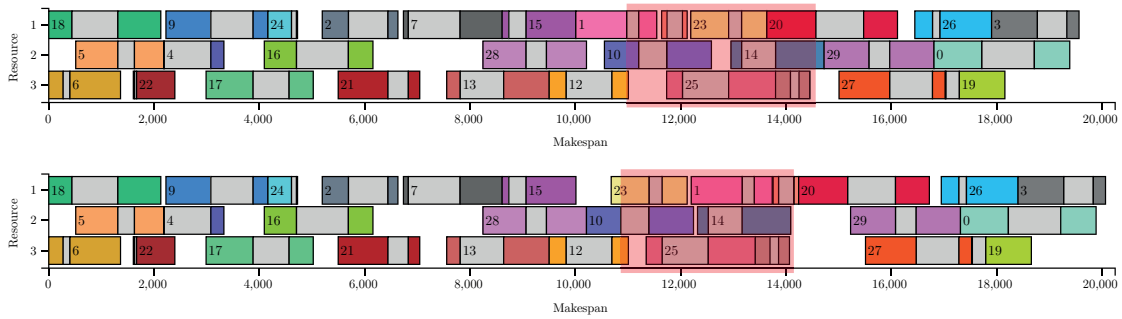



Figure 5.12: Local optimum obtained with $\mathcal{N}_B$, before and after it was perturbated with $\mathcal{N}^S_{\text{CP}}$.

## 5.7 Variable Neighborhood Search for the JSOCMSR

In the previous sections, different components of a VNS for the JSOCMSR were presented. This variety made it necessary to devise a generic framework based on appropriate abstractions that allows to combine those components in almost arbitrary ways. In the following section, we will briefly present the basic components of this framework. The neighborhoods to be selected in the GVNS are presented in Chapter 8, since a more detailed analysis on the structure of the induced landscapes is conducted beforehand.

The VNS framework devised for the JSOCMSR basically consists of a GVNS scheme, as introduced in Section 3.3 and shown in Algorithm 5.4, incorporating the *JSOCMSR_VND*, as shown in Algorithm 5.5 as intensification subprocedure. In general, the framework provides appropriate abstractions for both shaking as well as intensification neighborhood structures. While shaking neighborhood structures, on the one hand, perform arbitrary perturbation on the incumbent solutions, intensification neighborhood structures allow to explore the respective neighborhoods. The exploration is based on iterators that traverse the entire neighborhood based on an internally defined pattern. Once an improving solution is obtained, the neighborhood specification can decide whether the traversal can be terminated, i.e. the search then follows a *first-fit* strategy, or whether the neighborhood evaluation should be continued, i.e. *best-fit* strategy. Moreover, the framework also allows neighborhood structures to incorporate *neighbor-filters*. These filters enable the search procedure to exclude certain neighbors during the neighborhood evaluation. In our VNS, this approach is used to reduce the size of neighborhoods, as described in Section 5.4.

Finally, the intensification procedure, as shown in Algorithm 5.5, also allows to specify the step function that selects subsequent neighborhood structures in the VND. This mechanism allows to implement alternative VND schemes, like pipe [DGS06] or cyclic [TMM+17] as presented in Section 3.3.

**Algorithm 5.4:** General Variable neighborhood Search for the JSOCMSR

**Input:** An initial solution $s_0$, a set of shaking neighborhood structures $\mathcal{N}^S_{i=1\ldots k_{\max}}$, a set of intensification neighborhood structures $\mathcal{N}^I_{i=1\ldots l_{\max}}$ and step functions $\text{Change}^S(s, s', k)$ and $\text{Change}^I(k, improvementFound)$.

**Output:** A (possibly improved) solution $s$.

```
1  s ← s_0;
2  repeat
3      k ← 1;
4      while k ≤ k_max do
5          s' ← Shake(s, N_k^S);
6          s'' ← JSOCMSR_VND(s', N^I, Change^I);
7          (s, k) ← Change^S(s', s'', k);
8      end
9  until stopping criterion satisfied;
10 return s;
```

**Algorithm 5.5:** JSOCMSR_VND

**Input:** An initial solution $s_0$, a set of intensification neighborhood structures $\mathcal{N}^I_{i=1\ldots l_{\max}}$, and a change step function $\text{Change}^I(k, improvementFound)$

**Output:** A (possibly improved) solution s.

```
1  l ← 0;
2  s ← s_0;
3  repeat
4      initialize(N_l^I, s);
5      improvementFound ← false;
6      while s' ∈ N_l^I(s) do
7          if isAllowed(s') then
8              if MS(s') < MS(s) then
9                  persist(s');
10                 improvementFound ← true;
11                 if canStop() then
12                     break;
13     end
14     if improvementFound then
15         s ← restore();
16     (terminate, l) ← Change^I(l, improvementFound);
17 until stopping criterion satisfied ∨ terminate;
18 return s;
```

CHAPTER 6

# Experimental Setup and Fundamental Experiments

After the introduction to more theoretical concepts in the previous chapters, we will now turn the focus onto an experimental study of the devised concepts. In the beginning of this study, this short chapter will start with the experimental setup containing the execution environment, instances and used performance measures. Afterwards, results of fundamental experiments studying performance aspects of solution evaluation schemes will be presented.

## 6.1 Environment

All experiments were performed on a computing cluster consisting of 16 machines, each with two 10 core *Intel Xeon E5-2640 v4* CPUs and a total of 160$GB$ of memory per node. The different VNS approaches as well as the different experiments to analyze the search landscapes were implemented in C++11 using *G++ 7.4.0* with *-Ofast* optimization level. For some experiments and tests the following libraries were used:

- IBM ILOG CPOptimizer (12.7.1): state-of-the-art constraint optimization library optimized for planning and scheduling problems. It was primarily used to solve baseline constraint programming models or in experiments with hybrid solution methods.
- Boost (1.58): general purpose C++ library, primarily used for input processing and parallelized evaluation of experiments
- OpenMP (G++ 7.4.0): parallelization library used for result processing.
- *popcount*: the popcount instruction (*__builtin_popcount*) was used to efficiently evaluate critical set hamming distance metrics for optimal solutions with 30 jobs.

## 6.2 Instances

As mentioned in Chapter 1, in this work we primarily focus on two different classes of instances from [HRB19]—*balanced* and *skewed* ones. *Balanced* instances, on the one hand, were generated in a way to distribute the jobs uniformly among the available resources. On the other hand, in *skewed* instances, a job is assigned to secondary resource 1 with a probability of 0.5, while all other secondary resources $r_i$, $i > 1$ have a probability of $1/(2m-2)$.

Balanced instances on the one hand, were generated in a way such that all three phases of a job, i.e. preprocessing, utilization of resource 0 and postprocessing were sampled from the discrete uniform distribution in interval $[0, 1000]$ ($[1, 1000]$ for the resource 0). On the other hand, in skewed instances an imbalance with respect to the job phases is introduced, laying the focus on the utilization of resource 0, which is sampled from the discrete uniform distribution in interval $[1, 2500]$, while pre- and postprocessing phases are again randomly sampled from $[0, 1000]$.

Both instance sets consist of instances with $n \in \{50, 100, 200, 500, 1000, 2000\}$, $m \in \{2, 3, 5\}$ with 50 sampled instances each. In the following, we will refer to the balanced and skewed instances as $B$ and $S$ instances respectively.

## 6.3 Performance Measures

As already discussed in Section 3.2.3, the performance of metaheuristic algorithms can be quantified in various ways. In this work, we primarily direct our focus on the solution quality, but also consider some temporal aspects later on. We compare our obtained results by the commonly used *optimality gap*

$$\%\text{-gap} = \frac{\text{MS} - \text{MS}^{\text{LB}}}{\text{MS}^{\text{LB}}} \cdot 100\% \tag{6.1}$$

which is a relative measure of the distance of the objective value to a given lower bound $\text{MS}^{\text{LB}}$. As lower bound we use the strongest bounds obtained by Horn et al. [HRB19].

## 6.4 Synchronization

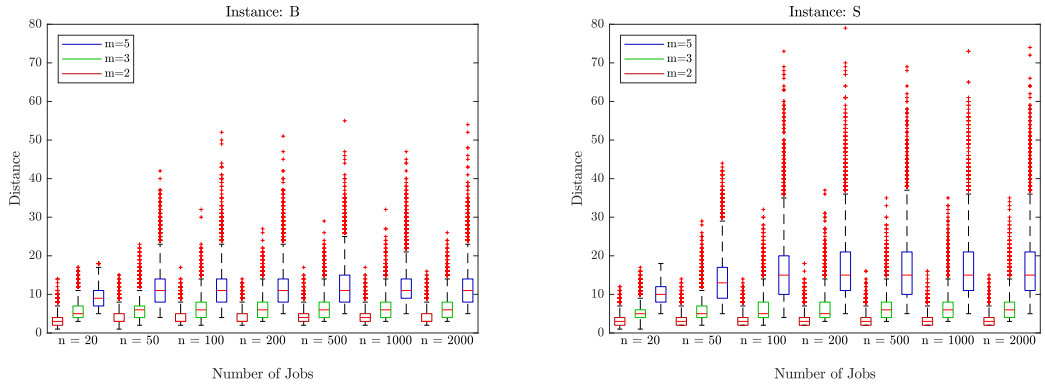
We start with a study of the practical applicability of the synchronization concept described in Section 5.2 that allows to evaluate the makespan of a neighboring solution quite efficiently. Figure 6.1 shows the average distance between the insertion point of the job and the synchronization point, i.e. the point where the re-evaluation can be prematurely terminated. Both *balanced* and *skewed* instances of different size were tested and $10^4$ random moves have been performed on 50 different instances for each of the given instance types. Based on those moves, the distance between the insertion destination and the synchronization point have been traced.

Generally, the *synchronization distance* tends to be not highly affected by the size of the instance. Wilcoxon rank sum tests have confirmed statistical significance for $\alpha = 0.01$ in some cases, but not to such an extent that it has a severe practical impact. In fact, this suggests that all resources are utilized throughout the entire schedule and hence the point where on all resources one job has aligned is found quite fast. For the studied instance types, the synchronization distance seems to be rather *constant* with respect to the number of jobs.

Furthermore, Figure 6.1 shows that the synchronization distance significantly increases with the number of secondary resources. An explanation for this behavior is that the increased number of secondary resources increases the size of the required *synchronization border* and consequently more steps are required to identify that border.

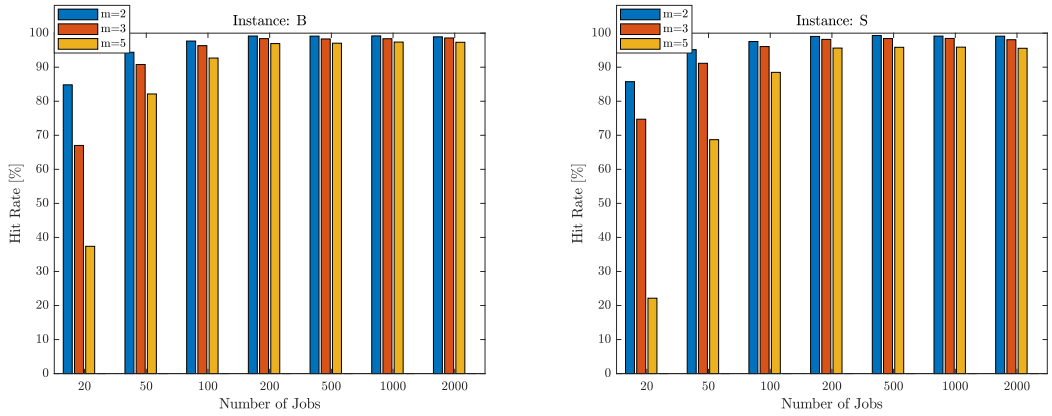
While experiments so far only showed the number of required steps in case the synchronization mechanism actually hits, we further investigated how often the mechanism can actually be applied in the given instances. Figure 6.2 shows the achieved hit-rate in $10^4$ random evaluations for balanced and skewed instances. For small instances, particularly with increasing secondary resources, the hit-rate drops down almost below 20%. An explanation for this poor hit-rate is that the end of the schedule was reached, before the synchronization border could be determined. On the other hand, however, with increasing number of jobs, Figure 6.2 shows a completely different behavior, where the hit-rate approaches almost 100%. For these instances, a randomly selected move is more likely to be far from the end of the schedule and therefore the synchronization border is found more frequently, before the end of the schedule is reached.

In conclusion, for the considered instance classes, the synchronization approach seems to be quite effective. On the one hand, synchronization seems to be quite resistant with respect to the increase in instance size, but also acceptable for increased number of secondary instances. Even more, for the considered instances, this approach showed a high hit rate for instances of non-trivial size. Due to these positive characteristics, building basic neighborhood moves on top of this technique seemed to be quite promising.




(a) (b)

Figure 6.1: Comparison of the synchronization distance in balanced and skewed instances with 20 to 2000 jobs. While the distance tends to increase with the number of resources $m$, it tends to remain constant in the number of jobs $n$.




(a) (b)

Figure 6.2: Hit-rate of the synchronization procedure of $10^4$ random evaluations averaged over 50 different instances for both balanced and skewed instances.

## 6.5 Evaluation Performance

The following section will contain a brief performance comparison of three different evaluation approaches:

1. Naive evaluation
2. Range-based re-evaluation (RB-EVAL)
3. Best-effort incremental evaluation (BEI-EVAL)

Figures 6.3a and 6.3b show the runtime behavior of insertion and exchange evaluation schemes with respect to increasing instance sizes. In the experiment, for each instance type and evaluation scheme, a total of 20000 single random moves have been performed on 50 different input instances. Due to the relatively low execution time of hundreds of nanoseconds, a sample has been considered an outlier and thus has been removed in case it has exceeded „more than three scaled median absolute deviations“ [1]. It can be observed that the naive and the range-based approaches do not scale well for neighborhoods of unbound move distance. In particular, for the range-based approaches a relatively high variance can be observed due to randomized selection of moves from the neighborhood in the benchmark. Despite this, for small and medium instances, the range-based approach seems to be not significantly slower than the incremental, so it may serve as a viable alternative due to less overhead for preparation. For the *incremental* approach, the tendency for a constant time evaluation remains and goes in accordance with results for the synchronization distance shown in Section 6.4. Again, it is important to note that devised evaluation schemes may not be that efficient in extremely imbalanced instances.

For neighborhood structures purely based on re-evaluation of subsequences in the solution, Figures 6.3c and 6.3d indicate an evaluation scheme constant in the size of the instance, but slightly growing with the size of the subsequence $k$. However, due to the rather marginal difference in $k$ used in the benchmark, this effect is only slightly indicated.

Finally, Figures 6.3d to 6.3f compare the execution times of evaluation schemes for different neighborhood structures. While for *insertion* and *exchange* neighborhoods, the BEI-EVAL approach is used, for *swap* and *inverse* RB-EVAL is applied. First, a clear increase in execution time with increasing number of secondary resources can be observed. While for instances with two secondary resources the evaluations of neighboring solutions can be performed in up to 400 ns, for $m = 3$ and $m = 5$ the execution time increases to 600 ns and 800 ns respectively.

[1] Samples were processed with the *MATLAB* function *rmoutliers*.

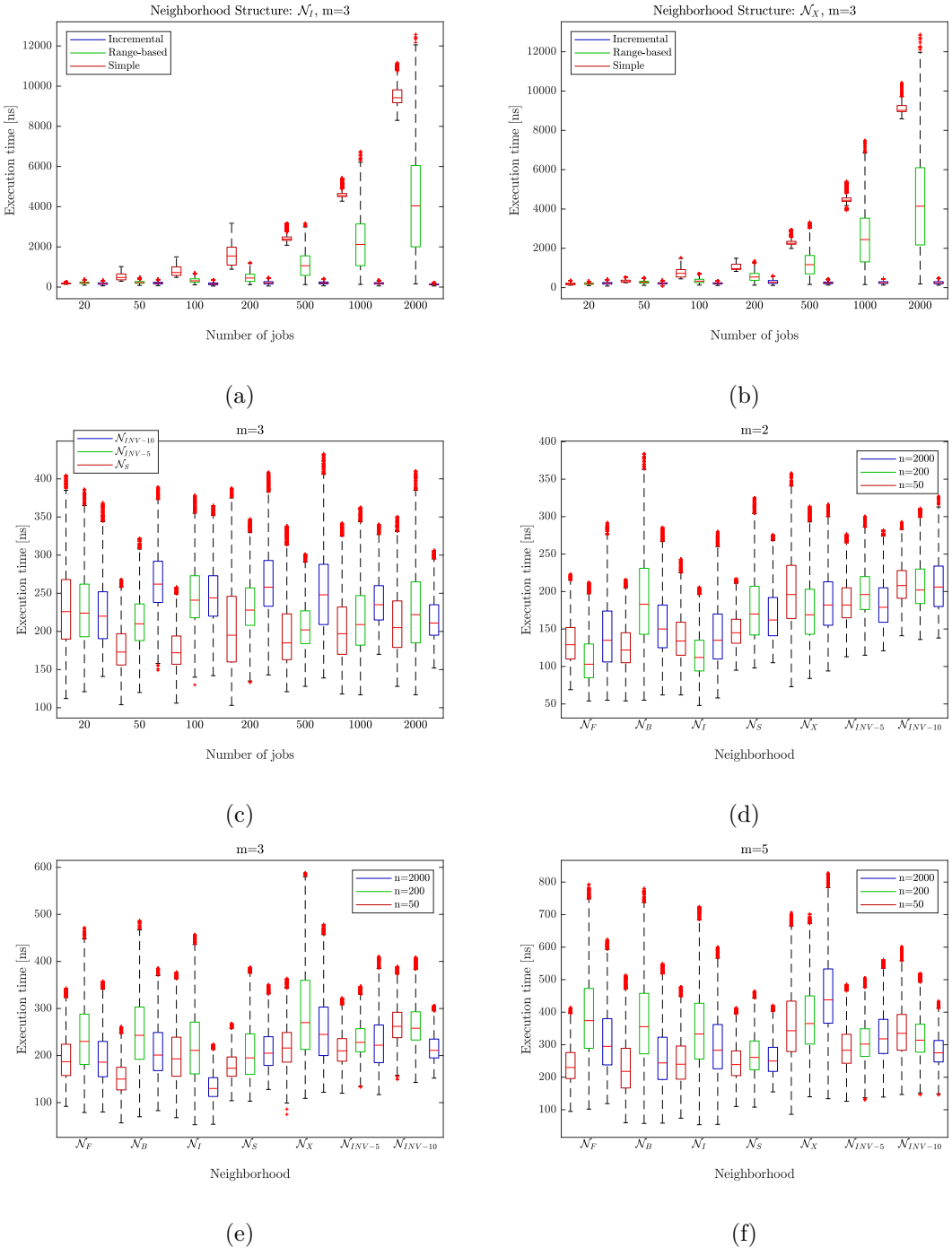



Figure 6.3: Comparison of the execution time of evaluation schemes of different moves. Figures 6.3a and 6.3b compare the execution time of three evaluation schemes of insertion and exchange moves with increasing number of jobs. Figure 6.3c compares the execution time of swap and inversion moves for instances with three secondary resources. Figures 6.3d to 6.3f compare execution times of the fastest evaluation scheme of all considered neighborhood structures for instances with $m = 2$, $m = 3$ and $m = 5$ for $n = 50$, $n = 200$ and $n = 2000$.

CHAPTER 7

# Search Space Structure and Landscape Analysis for Instances of the JSOCMSR Problem

In this chapter we try to take a closer look at the structure of search spaces and characteristics of *good* solutions. We start with a study on the ruggedness of different search landscapes in Section 7.1. In Section 7.2, we analyze characteristics of local optima induced by different neighborhood structures. Although good local optima are indeed desirable, the primarily goal is to find globally optima solutions. To this end, we try to investigate properties of global optima of the JSOCMSR problem in Section 7.3.

In general, studying landscapes of optimization problems is a non-trivial task. As explained in Chapter 4, there do exist plenty of different techniques. However, the size of the instances and the exponentially sized search spaces, make it difficult to actually conduct such studies with instances of reasonable size due to the relatively large number of repetitions required for statistical stability. Thus, the following chapter primarily concentrates on small to medium sized instances, for which extensive landscape studies become more feasible. Although findings in those instances do not necessarily hold true for larger instances, their similar nature due to the very same randomized generation procedure makes such an assumption more plausible.

## 7.1 Ruggedness of the Landscape

As already described in more depth in Section 4.3, the ruggedness describes a landscape induced by a neighborhood structure with respect to the fitness correlation between adjacent solutions. If this correlation is high, the landscape is rather smooth and the search can follow a long, gradually improving path towards a local optimum. A low correlation, on the other hand, indicates that the search encounters steep gradients and usually reaches local optima quite fast. Obviously, finding local optima fast, especially if they are of high quality, may be beneficial on the short term. However, those structural features in the landscape may also be of misleading nature and hence may complicate escaping from suboptimal regions in the search space.

To quantify the ruggedness of the search landscapes, an approach based on the autocorrelation function $r(d)$, see Definition 4.4 in Section 4.3, was used. Starting from an initial solution, a random walk of $10^6$ steps through the search space with a given move operator was performed for instances with $n = 100$. The sequence of objective values was then treated as a time series and analyzed by applying the *autocorrelation* function. To achieve statistical stability, 5 random walks were applied for 50 different instances of each instance class. The autocorrelation was calculated with the *MATLAB* function *autocorr* with a maximum *lag* of 10. Figure 7.1 shows the correlation length derived from the autocorrelation of random walks in instances with 100 jobs.

At a lag of 1, the correlation of objective values between adjacent solutions can be seen. This value may be interpreted as an immediate indicator for the smoothness of the landscape. Not very surprising, in all plots in Figure 7.1 the swap neighborhood shows a relatively *smooth* landscape, followed by an inversion neighborhood structure of three consecutive jobs (i.e. exchange with a distance of 2). A plausible reason for this seems to be the rather limited number of structural changes in the solution induced by the move. For more *destructive* move operators, like exchange or the inversion of large subsequences, the correlation length is significantly lower, indicating a more rugged landscape.
With an increasing number of lags, the correlation length of all move operators seems to convergence towards a common value, obviously due to the fact that with an increasing number of random moves, more and more structural components of a solution wear out until little to no correlation can be observed.

Interestingly, when looking again at the correlation lengths induced by neighborhood structures $\mathcal{N}_S$, $\mathcal{N}_{INV-3}$ and $\mathcal{N}_X$ moves, a significant high difference can be observed, even though the moves are very similar in nature. The data suggests that the length of the move has a significant impact on the ruggedness of the landscape, most likely due to the sequential construction of a solution, i.e. shorter moves only affect closer regions of a solution, while longer moves destroy more structure and hence may have a higher impact on the solution's makespan. Figure 7.2 illustrates this effect by comparing the correlation length of insertion and exchange moves with limited range. First, the ruggedness of landscapes induced by the insertion neighborhood structure $\mathcal{N}_I$ tends to be less affected by the range, than in $\mathcal{N}_X$, likely due to the more destructive nature of exchange moves.

Moreover, in Figure 7.2 can in general be seen that the move range has indeed a significant effect on the smoothness of the neighborhood. So for particularly low range limits, the landscape's smoothness tends to be rather high, while it soon converges towards a common value for limits greater than 20.

In conclusion, the neighborhood structure has a crucial impact on the search landscapes' ruggedness. While structure preserving moves, like in $\mathcal{N}_S$, or move operations within short ranges tend to induce a rather smooth landscape, moves in exchange or inversion neighborhood structures, or in general more destructive moves with respect to the structure of a solution, tend for more rugged landscapes. Finally, the obtained correlation lengths in Figure 7.1 and Figure 7.2 suggest that *balanced* instances tend to induce smoother landscapes for the considered neighborhood structures than *skewed* instances.

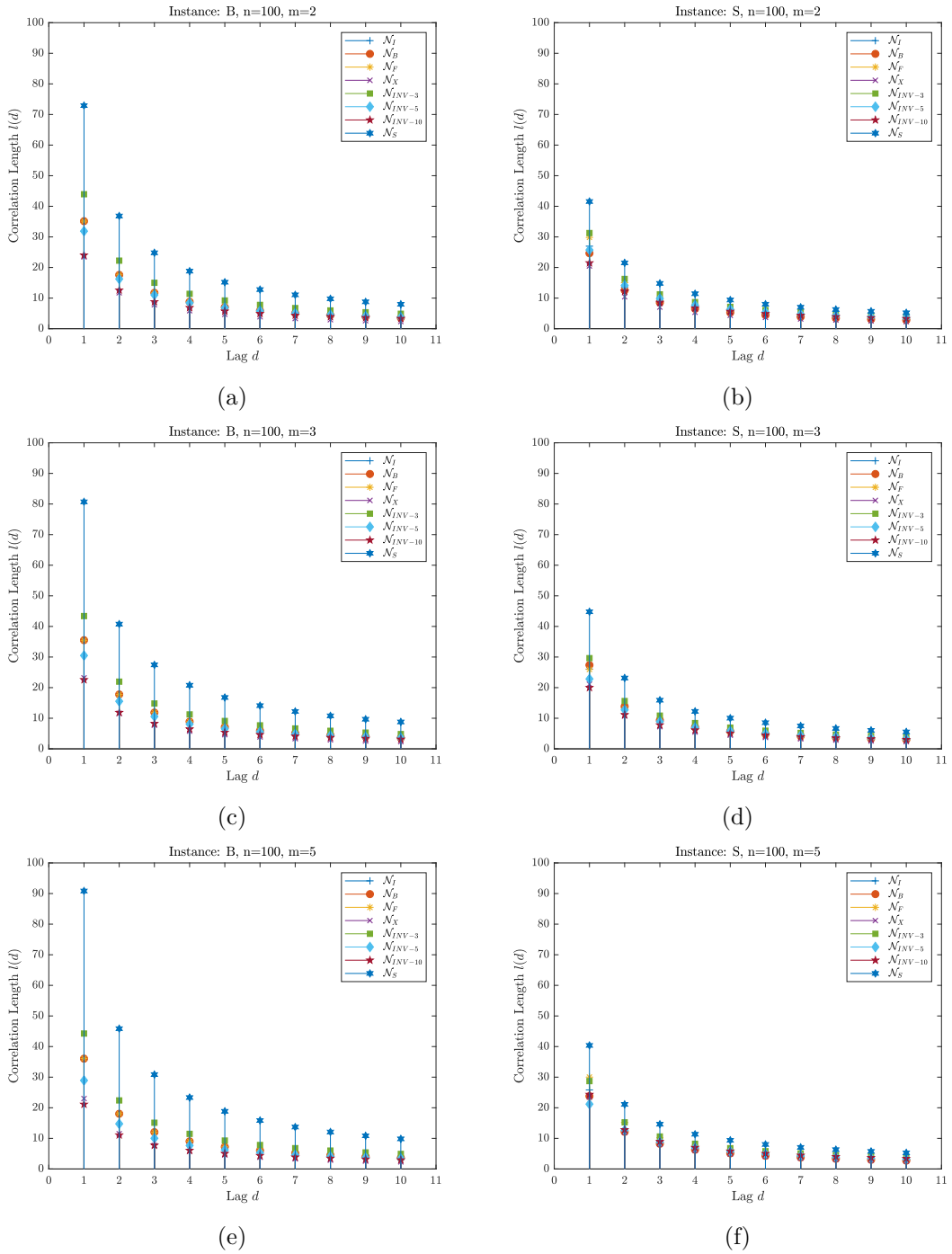



Figure 7.1: Correlation length of balanced and skewed instances with 2, 3 and 5 secondary resources and 100 jobs.

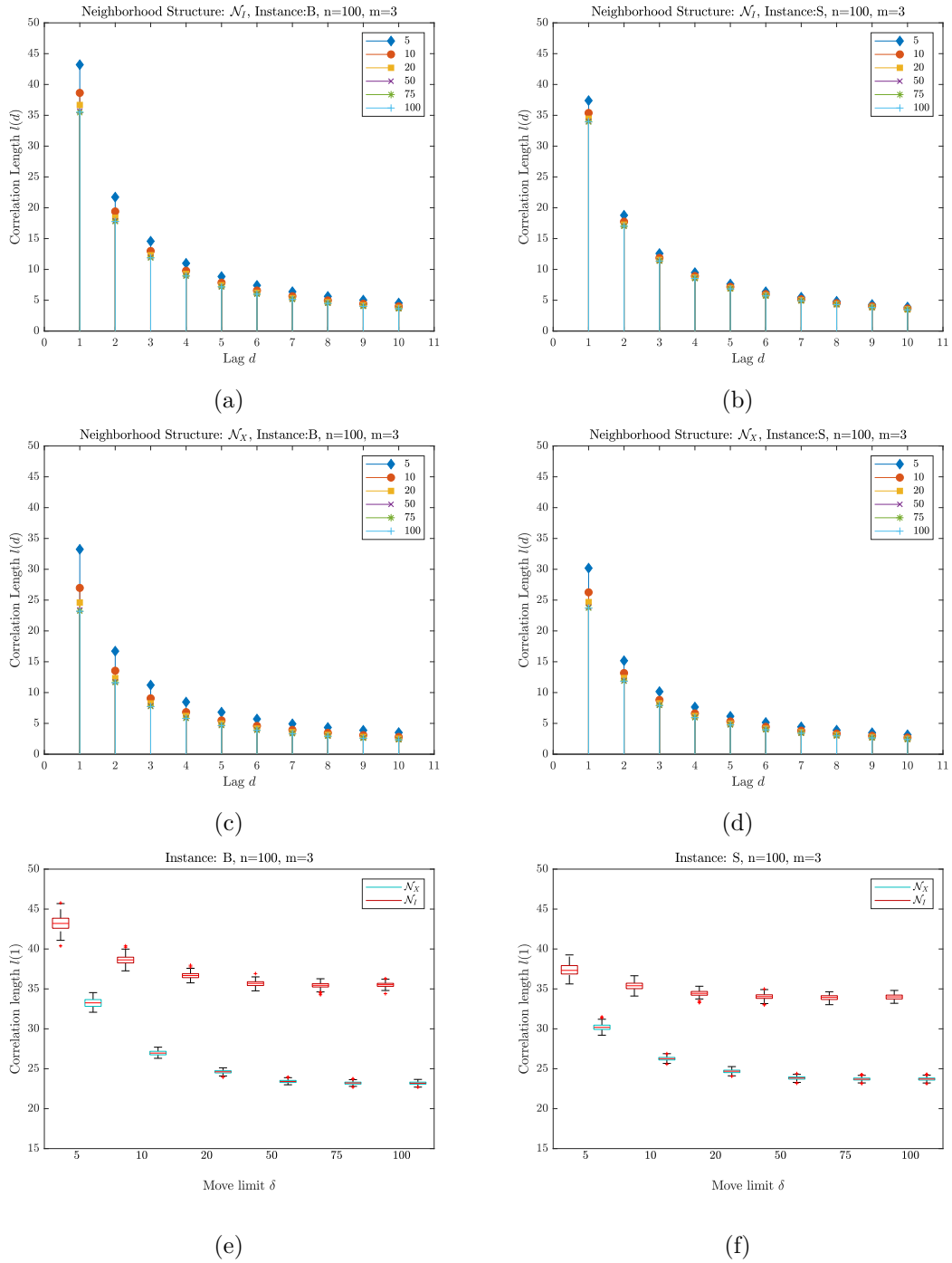



Figure 7.2: Correlation length of balanced and skewed instances with 2, 3 and 5 secondary resources, 100 jobs and 6 different move-range limitations.

## 7.2 Characteristics of Local Optima

The following section compares different aspects of locally optimal solutions obtained by local search procedures based on different neighborhoods structures. By studying certain characteristics of those optima, the goal is both to gain a basic understanding of the induced search landscapes, but also to compare different performance characteristics of the considered neighborhood structures. These results are then used to devise appropriate neighborhood combinations for the VND intensification phase in our VNS.

To this end, an experiment was conducted where local searches with 7 different neighborhood structures were performed on both *balanced* and *skewed* instances. Overall, 2500 local searches were performed for 50 different instances each. Further, different instance sizes have been considered, however, due to the vast amount of executions and the implied total runtime, the maximum number of jobs considered was $n = 100$.

### 7.2.1 Quality

Figure 7.3 compares the optimality gap of local optima obtained by a *best-fit* local search procedures starting from random solutions. Throughout all instance types a significant difference can be observed between neighborhood structures $\mathcal{N}_I$ and $\mathcal{N}_X$ on the one hand, and $\mathcal{N}_S$ and $\mathcal{N}_{INV}$ on the other. A reasonable explanation for this behavior may be the difference in the size of the neighborhoods with $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$ respectively.

More interestingly are indeed Figures 7.3c and 7.3d, focusing only on the optimality gap of local optima produced by $\mathcal{N}_F$, $\mathcal{N}_B$, $\mathcal{N}_I$ and $\mathcal{N}_X$ neighborhoods. In these plots, the modified scale allows to observe more clearly the optimality differences induced by the different neighborhood structures. In a first observation, a clearly different distribution between balanced and skewed instances can be observed. For the balanced instances, all neighborhood structures tend to struggle with instances with three secondary resources, where some outliers even fall above an optimality gap of 16%. For instances with $m = 2$ and $m = 5$, however, local searches already seem to provide high quality results, in particular for neighborhood structures $\mathcal{N}_I$ and $\mathcal{N}_X$. For skewed instances on the other hand, in instances with two secondary resources it tends to be more difficult to reach local optima of high quality. Later on, in Section 7.3 we show that characteristics like resource utilization imbalance or the relative utilization of the common resource may be among the factors affecting the hardness of a solution.

Equally important, at least from a performance perspective, is the difference between one-sided insertion neighborhood structures $\mathcal{N}_F$, $\mathcal{N}_B$ and the unrestricted variant $\mathcal{N}_I$. For our implementation we showed in Section 6.5 that the one-sided neighborhood structures $\mathcal{N}_F$ and $\mathcal{N}_B$ are advantageous in terms of evaluation performance. Although these neighborhoods are of smaller cardinality, the boxplots show only a small difference with quite a lot of outliers, suggesting that the improvement in terms of quality may not be of significant nature. To this end, for $n = 100$ and an overall sample size of 125000 local optima, pairwise *Wilcoxon Ranksum Tests* with $\alpha = 0.01$ have been performed to determine statistical significance of the medians.

The tests showed that one-sided insertions neighborhood structures $\mathcal{N}_F$ and $\mathcal{N}_B$ do not significantly differ (corresponding p-values between 0.5 and 1), while the optimality gap to $\mathcal{N}_I$ and $\mathcal{N}_X$ is indeed significant (each with a corresponding p-value $< 0.00001$). This suggests that the additional cost in terms of computation time is beneficial with respect to the quality. Between $\mathcal{N}_I$ and $\mathcal{N}_X$ also a clearly significant difference could be observed with p-values between 0 and $9.7058e^{-6}$.

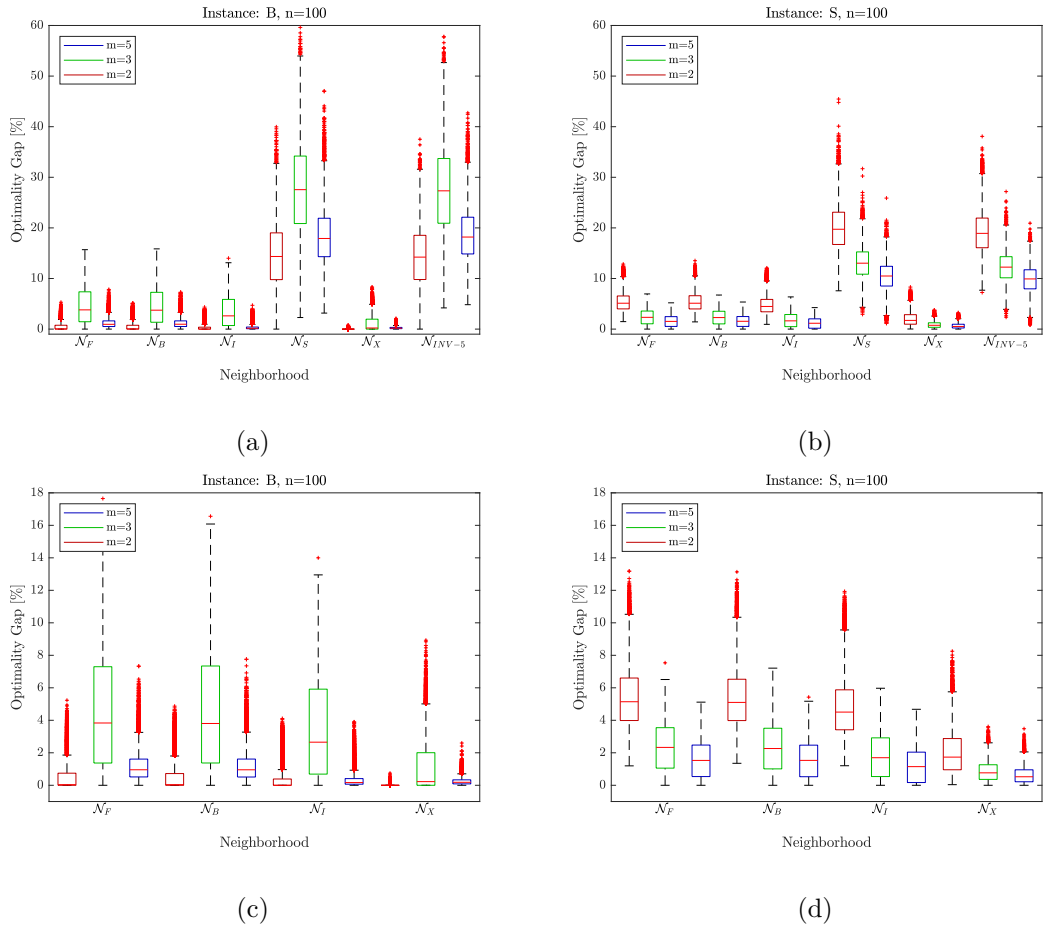
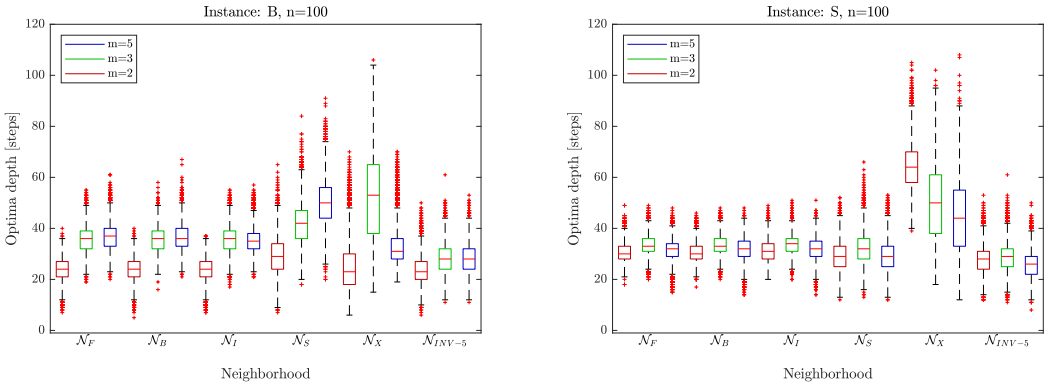



Figure 7.3: Comparison of the average optimality gap (see Equation (6.1)) of local optima obtained by local search procedures with neighborhood structures $\mathcal{N}_F$, $\mathcal{N}_B$, $\mathcal{N}_I$, $\mathcal{N}_X$, $\mathcal{N}_S$, $\mathcal{N}_{INV-5}$.




(a) (b)

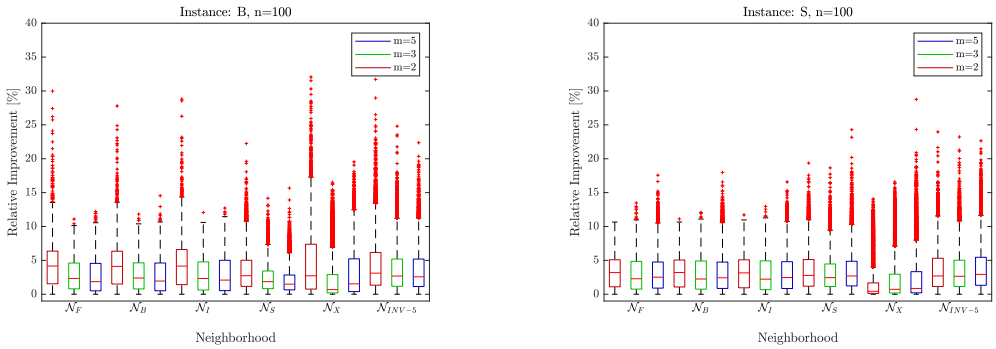
Figure 7.4: Comparison of the depth of local optima obtained by local search procedures based on neighborhood structures $\mathcal{N}_F$, $\mathcal{N}_B$, $\mathcal{N}_I$, $\mathcal{N}_X$, $\mathcal{N}_S$, $\mathcal{N}_{INV-5}$.

### 7.2.2 Depth

Figure 7.4 compares the depth, i.e. the number of steps in the search path from the initial solution to the local optimum from the previous experiment for $n = 100$.

Generally, in the studied instances local optima tend to be rather shallow and tests have shown that they can be reached approximately after $n$ local search iterations. Further, local search based on exchange neighborhoods structures $\mathcal{N}_X$ tend to induce significantly deeper and superior optima. Figures 7.5a and 7.5b show the distributions of the relative improvements of single steps in the local search. This plot indicates whether in the walk through the search space certain moves account for a majority of the improved makespan, thus making it crucial to select them during neighborhood exploration.

Moreover, it can be observed that the distribution is rather similar across the neighborhood structures, although for some a one-sided levelling out distribution with many outliers suggests that some particular moves in the local search induce a significantly high makespan improvement. Interesting behaviors can in particular be observed for $\mathcal{N}_S$ and $\mathcal{N}_{INV}$ neighborhood structures. For $\mathcal{N}_S$, relatively deep local optima can be observed. However, the previous section showed that they are in general of rather poor quality, making them only appropriate for huge instances due to their computational efficiency. Inversion neighborhoods seem to have rather shallow local optima, although the relative performance improvement with a relatively high number of outliers suggests that certain moves account for rather large changes in the makespan.




(a) (b)

Figure 7.5: Comparison of the relative improvement of a single step in the search for local search procedures based on neighborhood structures $\mathcal{N}_F$, $\mathcal{N}_B$, $\mathcal{N}_I$, $\mathcal{N}_X$, $\mathcal{N}_S$, $\mathcal{N}_{INV-5}$.

### 7.2.3 Complementing Neighborhoods

The previous section showed that local optima obtained by local search procedures with single neighborhood structures already obtain rather good solutions, sometimes even close to an optimality gap of approximately 2%. However, it was also shown that local optima also tend to be rather shallow and local search procedures tend to be trapped in local optima quite fast.

The goal of this section is to investigate how the different neighborhood structures can complement each other in a VND-based optimization procedure. To this end, from each of the 2500 local optima of each instance class of the previous experiment, 500 optima were selected randomly. This set of local optima was then used as input to an evaluation of the respectively other neighborhood structures, in order to determine whether they can complement each other in a VND like procedure.

Table 7.1 shows the percentage of 150.000 local optima, where the respective neighborhood structures (listed in the columns) could obtain at least one improving neighbor. Indeed, this benchmark does not investigate how long those subsequent trajectories with the new neighborhood structure would then guide the search into a new local optimum, however, we show that some neighborhood structures complement each other and allow to escape at least to some extent from local optima obtained by others.

First, the hierarchical relationships between related neighborhood structures is clearly observable. While the one-sided insertion in $\mathcal{N}_F$ and $\mathcal{N}_B$ can obviously not improve their generalization $\mathcal{N}_I$, the high percentage of improved neighbors between $\mathcal{N}_F$ and $\mathcal{N}_B$ shows that neighborhood moves in only one direction may be insufficient. A similar hierarchical relationship is shown for $\mathcal{N}_X$ and $\mathcal{N}_S$.

More interestingly are indeed the improvement rates between insertion-based neighborhood structures $\mathcal{N}_F$, $\mathcal{N}_B$, $\mathcal{N}_I$ and the exchange neighborhood structure $\mathcal{N}_X$. With an

| LO \ Impr. | $\mathcal{N}_F$ | $\mathcal{N}_B$ | $\mathcal{N}_I$ | $\mathcal{N}_X$ | $\mathcal{N}_S$ | $\mathcal{N}_{INV-5}$ |
|---|---|---|---|---|---|---|
| $\mathcal{N}_F$ | - | 84.346 | 84.346 | 90.912 | 0.000 | 60.464 |
| $\mathcal{N}_B$ | 84.252 | - | 84.252 | 90.868 | 0.000 | 60.461 |
| $\mathcal{N}_I$ | 0.000 | 0.000 | - | 80.498 | 0.000 | 48.847 |
| $\mathcal{N}_X$ | 61.520 | 61.966 | 68.969 | - | 0.000 | 2.041 |
| $\mathcal{N}_S$ | 99.995 | 99.995 | 99.995 | 99.995 | - | 99.724 |
| $\mathcal{N}_{INV-5}$ | 99.994 | 99.994 | 99.994 | 99.994 | 99.970 | - |

Table 7.1: Percentage of local optima obtained by the neighborhood structures listed in the rows (LO) that could be improved at least once by neighborhood structures shown in the columns (Impr.).

improvement rate approximately between 60% and 80%, this clear shows how local optima obtained by one neighborhood structure are likely to be escaped by other the respectively other neighborhood structure. Based on these findings, it may be a reasonable option to combine those neighborhoods in a VND-based improvement phase, where insertion-based neighborhood structures are used to find local optima fast and the more expensive *exchange* moves are then used in later phases for diversification and escaping mechanisms.

For $\mathcal{N}_S$, almost any local optimum could be improved by other neighborhoods, so either swap may be used as innermost neighborhood in a VND or not at all due to the rather poor quality of local optima.

The previous sections also compared different aspects of inversion-based neighborhood structures $\mathcal{N}_{INV-k}$ and generally came to the conclusion that despite its evaluation efficiency, the neighborhood structure is not very natural to the JSOCMSR and local optima tend to be of poor quality. However, most likely due to the completely different nature of the move, Table 7.1 clearly shows that inversion moves are able to improve a significant amount of other local optima. Therefore, it may be a reasonable approach to still employ inversion neighborhood structures in VND schemes for the sake of diversity, even though they perform poorly in standalone local search procedures.

## 7.3 Characteristics of Global Optima

Characteristics of global optima can provide valuable insight on the ingredients good solutions are made of and properties like, the number and the distribution of global optima may already act as a rough hardness indicator.

When looking for reappearing structures in solutions, it is often useful to have a large number of global optima available. While for well-studied problems, like the *TSP*, optimal solutions for standard instances are often known, they are hardly available for fairly new problems as the JSOCMSR. It is therefore inevitable, to aggregate optima in numerous

| | | % found optima | | | % distinct optima | | | %-gap | |
|---|---|---|---|---|---|---|---|---|---|
| type | $m$ | $\mu$ | $\sigma$ | min | $\mu$ | $\sigma$ | min | $\mu$ | $\sigma$ |
| B | 2 | 100.0 | 0 | 1 | 1 | <0.01 | 1 | 0 | 0 |
| B | 3 | 97.10 | 0.139 | 0.117 | 0.980 | 0.117 | 0.196 | 0 | 0 |
| B | 5 | 100.0 | <0.01 | 0.979 | 1 | 0 | 1 | 0 | 0.001 |
| S | 2 | 98.10 | 0.078 | 0.491 | 0.950 | 0.189 | 0.136 | 0.003 | 0.007 |
| S | 3 | 99.90 | <0.01 | 0.960 | 0.999 | <0.01 | 0.930 | 0.002 | 0.008 |
| S | 5 | 95.40 | 0.177 | 0.224 | 0.939 | 0.234 | 0.023 | 0.000 | 0.001 |

Table 7.2: The percentage of obtained globally optimal solutions, the percentage of distinct solutions in this set and the average optimality gap as measure for solution optimality. The mean and the standard deviation are denoted by $\mu$ and $\sigma$ respectively.

experiments and stick to instances of moderate size. But even for such instances, it is often not even possible to obtain proven optimal solutions and best-known solutions have to suffice.

### 7.3.1 Number and Distribution of Global Optima

For the JSOCMSR problem, a large number of (likely) globally optimal solutions was obtained by repeatedly starting a VNS from random initial solutions for instances with 30 jobs. On the one hand, we claim that 30 is a reasonable choice since the instances are sufficiently large to potentially contain repeatedly occurring patterns (e.g. interleaving of jobs on different resources), while still being small enough to be able to obtain (nearly) optimal solutions in reasonable time. For both, the skewed and the balanced instances, 12500 repetitions of the VNS were performed for 50 different instances each. While the VNS can in general not proof a solutions optimality, the tight lower bound introduced by Horn et al.[HRB17] was used to confirm optimality in most cases. In cases, where the lower bound did not suffice, other search methodologies, in particular *ILOG CP Optimizer* has been applied to strengthen the optimality claim.

Table 7.2 shows the percentage of found optima in 12500 runs, the percentage of distinct global optima as well as the optimality gap, each with the mean $\mu$ and the standard deviation $\sigma$. It is shown that the number of global optima found is in general relatively high. For balanced instances, in particular with two or five secondary resources, almost any of the $50 * 12500$ runs found a globally optimal solution. For balanced instances with three secondary resources, similar results have been obtained, except for two particularly hard instances, where the number of found optima dropped to 11.7% and 60.6% respectively.

To analyze this surprisingly large number of distinct global optimal solutions, a more detailed look into some static instance metrics has to be taken. First,

$$\overline{r_0^{\text{util}}} = \frac{1}{n} \sum_{j \in J} \frac{p_j^0}{p_j} \tag{7.1}$$

| type | $m$ | $\overline{r_0^{\text{util}}}$ | $\overline{r}^{\text{imb}}$ |
|---|---|---|---|
| B | 2 | 0.3328 ($\pm$ 0.0300) | 0.7276 ($\pm$ 0.1690) |
| B | 3 | 0.3256 ($\pm$ 0.0241) | 0.7184 ($\pm$ 0.1038) |
| B | 5 | 0.3346 ($\pm$ 0.0253) | 0.6201 ($\pm$ 0.1103) |
| S | 2 | 0.5477 ($\pm$ 0.0313) | 0.7806 ($\pm$ 0.1597) |
| S | 3 | 0.5606 ($\pm$ 0.0282) | 0.5839 ($\pm$ 0.1342) |
| S | 5 | 0.5648 ($\pm$ 0.0261) | 0.4459 ($\pm$ 0.1147) |

Table 7.3: Metrics describing the relative utilization of the common resource 0 and the imbalance in utilization of the secondary resources in different balanced and skewed instances.

quantifies the average relative utilization of the common resource 0 over all jobs in the instances. Hence, when $\bar{r}_0^{\text{util}}$ tends towards 0, secondary resources become more dominant. On the other hand, when $\overline{r_0^{\text{util}}}$ is close to 1, the common resource dominates.

The second metric,

$$\overline{r}^{\text{imb}} = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \min(p_i^{\Sigma}/p_j^{\Sigma}, p_j^{\Sigma}/p_i^{\Sigma}) \tag{7.2}$$

tries to quantify the imbalance in the utilization of the secondary resources. Based on the overall time

$$p_r^{\Sigma} = \sum_{j \in J_r} p_j \tag{7.3}$$

resource $r$ is utilized, the metric averages the minimal, pairwise ratios of the overall resource utilization. So in case the instance is perfectly balanced, the metric indicates this with a value close to 1. Otherwise $\overline{r}^{\text{imb}}$ tends towards 0.

Table 7.3 contains average values over all considered instances for both metrics. Compared to skewed instances, balanced instances have a relatively low utilization of the common resource 0, which may result in a reduced contention about resource 0 between jobs on different secondary resources. Further, the utilization of the secondary resources tends to be rather imbalanced. For balanced instances with two secondary resources, the lower utilized resource only makes up $\approx 72\%$ of the more dominant resource. This, together with the rather low utilization of the common resource implies that the makespan is primarily determined by the dominant resource and jobs from the other resource tend to only fill up gaps. In particular, the experiment showed that for balanced instances, 84.5% of the jobs from the less dominant resources are placed exactly between two jobs of the dominant resource and therefore do rarely content with respect to resource 0. This behavior is illustrated in Figure 7.6 showing a global optimal solution of a balanced instance with 20 jobs.

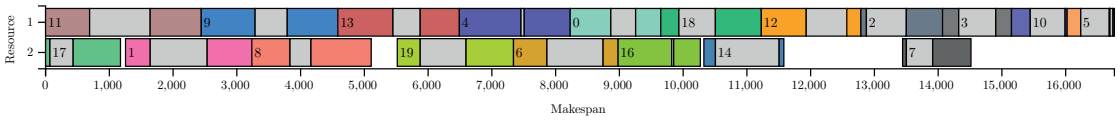



Figure 7.6: Global optimum of a balanced instance with two secondary resources.

In contrast, for skewed instances the number of found global optima is significantly lower. One reason for this could be the increased utilization of the common resource compared to the overall processing time, which in turn results in an increased contention between jobs on different secondary resources. Figure 7.7 shows an illustrative example of this behavior. The increased relative utilization of the common resource delays jobs on other resources, resulting in gaps in the utilization on the secondary resource, which in turn inherently increase the makespan.

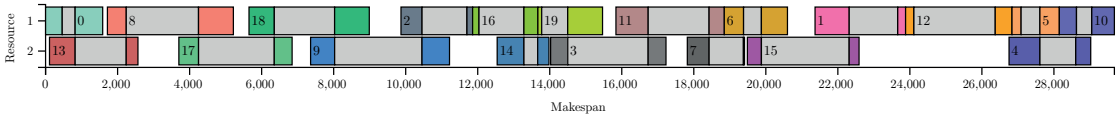



Figure 7.7: Global optimum of a skewed instance with two secondary resources.

More interestingly than the number of optima found in the experiment is the actually the number of distinct global optima. Table 7.2 shows that each run for balanced instances with two or five secondary resources has obtained a different globally optimal solution. As already described previously, it seems that for those instances the dominant resources effectively define the makespan and jobs of less utilized resources fill up idle times. As a result of this imbalance, the submissive resources suffer from some kind of fragmentation, where the secondary resource remains idle for longer phases. In these phases, similar jobs on the same resource can often be swapped, without affecting the makespan at all. Again, this effect can be observed in Figure 7.6.

Finally, Table 7.4 compares the average pairwise distances between the set of optimal solutions and a random population. Distance metrics were chosen to be appropriate for *exchange* and *insertion* neighborhoods, as described in Section 4.2. Generally, it can be seen that the average distance of the random population tends to be significantly higher than for the set of globally optimal solutions, suggesting them to be concentrated in a closer area of the search space than the random population being expected to be evenly distributed in the search space. Nevertheless, the difference between the random population and the set of optima still tends to be relatively small, suggesting that globally optimal solutions share some solution components, but are still widely distributed in the search space.

| | | Optimal Solutions | | Random Solutions | |
|---|---|---|---|---|---|
| type | $m$ | $d_{\mathcal{C}}$ | $d_{\mathcal{U}}$ | $d_{\mathcal{C}}$ | $d_{\mathcal{U}}$ |
| B | 2 | 0.893 (± 0.003) | 0.740 (± 0.002) | 0.897 (± 0.053) | 0.746 (± 0.042) |
| B | 3 | 0.885 (± 0.023) | 0.732 (± 0.022) | 0.897 (± 0.053) | 0.744 (± 0.042) |
| B | 5 | 0.825 (± 0.038) | 0.679 (± 0.034) | 0.897 (± 0.053) | 0.745 (± 0.042) |
| S | 2 | 0.855 (± 0.060) | 0.639 (± 0.076) | 0.897 (± 0.053) | 0.744 (± 0.042) |
| S | 3 | 0.835 (± 0.045) | 0.668 (± 0.028) | 0.896 (± 0.053) | 0.746 (± 0.043) |
| S | 5 | 0.810 (± 0.072) | 0.648 (± 0.073) | 0.897 (± 0.054) | 0.744 (± 0.042) |

Table 7.4: Comparison of average normalized distances in two sets of solutions in different instance classes. One set consists of global optima, the other of random solutions.

### 7.3.2 Critical Jobs

In Section 5.4.3 it was shown that each instance consists of a set of critical jobs that define the makespan, the other jobs do not have any immediate impact on the makespan.

In the literature, the backbone of a solution is usually described as some sort of structure that is frequently encountered in a set of good or optimal solutions [Wat10]. Identifying the solution backbone for an instance or even for a class of instances, provides an in-depth insight into the nature of the considered solutions. Although a thorough backbone analysis for instances of the JSOCMSR is beyond the scope of this work, this section aims to study solutions with respect to properties of their critical set, which may be considered as some kind of solution backbone.

For all global optima of the experiment, the set of critical jobs has been determined and three different metrics have been defined and derived to describe those critical sets. The first metric to be analyzed is the average normalized hamming distance $\bar{d_{\mathcal{H}}}$ of binary strings indicating whether a job is part of the critical set or not, i.e. 1 or 0 respectively. In case the average distance is close to 1 in this scenario, the set of critical jobs tends to be rather diverse, while values close to 0 indicate a rather stable set of critical jobs. The second metric is the average number of distinct critical sets and is denoted as $\bar{\#_{\mathcal{C}}}$. Hence, if the average hamming distance $\bar{d_{\mathcal{H}}}$ is close to 0, it is expected that the number of distinct critical sets also tends to be rather low. Finally, the third metric considered in this study is the average cardinality of the critical set, denoted as $\bar{|\mathcal{C}|}$. Since we consider instances of 30 jobs, the value has apparently a range between 0 and 30.

Table 7.5 shows the metrics averaged over all globally found optima of the six different instance classes. Compared to the maximum hamming distance of a binary string of size 30, the average distance shown for all instance classes is rather low. Hence, for a majority of optimal solutions, it seems that the very same jobs constitute the set of critical jobs. In particular for imbalanced instances with two or three resources the average distance is remarkably low, indicating that the set of critical jobs is constituted of a particular set of dominant jobs.

| type | $m$ | $\bar{d_{\mathcal{H}}}$ | $\bar{\#_{\mathcal{C}}}$ | $\lvert\bar{\mathcal{C}}\rvert$ |
|---|---|---|---|---|
| B | 2 | 0.001 ($\pm$ 0.002) | 69.30 ($\pm$ 81.54) | 17.06 ($\pm$ 1.94) |
| B | 3 | 0.002 ($\pm$ 0.002) | 71.27 ($\pm$ 76.68) | 12.55 ($\pm$ 1.33) |
| B | 5 | 0.010 ($\pm$ 0.008) | 107.2 ($\pm$ 61.86) | 26.02 ($\pm$ 7.70) |
| S | 2 | 0.003 ($\pm$ 0.006) | 5.53 ($\pm$ 5.24) | 23.77 ($\pm$ 3.06) |
| S | 3 | 0.002 ($\pm$ 0.003) | 5.16 ($\pm$ 4.11) | 27.71 ($\pm$ 3.34) |
| S | 5 | 0.004 ($\pm$ 0.012) | 4.07 ($\pm$ 3.12) | 27.61 ($\pm$ 3.54) |

Table 7.5: Critical set metrics averaged over all globally optimal solutions obtained for six different instance classes with $n = 30$.

More interestingly is the number of distinct critical sets. As can be seen, for the considered global optima, the metric falls into a range between approximately 4 and 110. Despite the given standard deviation and taking into account the theoretical maximum of up to 12500 distinct critical sets, the data suggests that a large set of (distinct) globally optimal solutions is actually constituted of the very same critical set.

Finally, the average size of the critical set may also provide valuable insight into this effect. For the instances with $n = 30$ jobs, the cardinality of the critical set varies between 12 and $\approx 28$. Taking now the *relatively* low standard deviation, the number of distinct global optima given in the previous section and even more possible symmetries due to jobs of similar size into account, this may indicate that *typically* each globally optimal solution is constituted of approximately the very same set of critical jobs. However, it is important to note that it is not clear whether this characteristic also holds true for larger instances. Table 7.6 compares the same metrics for local optima obtained by a VND consisting of *exchange* and *insertion* neighborhoods. In total, for each of the 50 different instance classes, 5000 local optima were obtained. Although the normalized average hamming distance $\bar{d_{\mathcal{H}}}$ is far from being comparable to the global optima shown previously, in some classes the instances still tend to consist of a set of dominant jobs that constitute the critical set already only in locally optimal solutions. Based on this finding, future approaches may exploit this to reduce instances to the most relevant jobs in a preprocessing step or recurrent phases to continuously refine the set throughout the search and subsequently try to find suitable gaps for the other jobs. With respect to the other two metrics, $\bar{\#_{\mathcal{C}}}$ and $\lvert\bar{\mathcal{C}}\rvert$ differ with respect to the global optima. On the one hand, $\lvert\bar{\mathcal{C}}\rvert$ tends to be rather similar between the global optima for $n = 30$ and the local optima for $n = 100$ and $n = 200$. This may be an indicator that imbalances with respect to the resource utilization also appears in larger instances. The number of distinct critical sets, on the other hand, is significantly higher despite the smaller sample size of 5000 for local optima and 12500 for global optima. A plausible explanation for this effect may both be the larger instance size, where small variations are more likely and the suboptimal nature of the local optima.

| type | $n$ | $m$ | $\bar{d_{\mathcal{H}}}$ | $\bar{\#_{\mathcal{C}}}$ | $\lvert\bar{\mathcal{C}}\rvert$ |
|---|---|---|---|---|---|
| B | 100 | 2 | 0.01 (± 0.018) | 324.34 (± 537.998) | 53.45 (± 3.600) |
| B | 100 | 3 | 0.13 (± 0.175) | 2226.62 (± 1985.712) | 43.78 (± 8.154) |
| B | 100 | 5 | 0.02 (± 0.037) | 955.46 (± 811.198) | 97.64 (± 9.235) |
| S | 100 | 2 | 0.07 (± 0.054) | 2868.66 (± 1715.838) | 81.75 (± 7.548) |
| S | 100 | 3 | 0.03 (± 0.025) | 1325.42 (± 1349.472) | 90.52 (± 10.158) |
| S | 100 | 5 | 0.02 (± 0.019) | 720.28 (± 926.006) | 89.98 (± 10.915) |
| B | 200 | 2 | 0.01 (± 0.015) | 613.48 (± 624.368) | 103.89 (± 5.446) |
| B | 200 | 3 | 0.16 (± 0.173) | 3197.74 (± 2016.495) | 86.11 (± 16.089) |
| B | 200 | 5 | 0.01 (± 0.002) | 1501.44 (± 223.532) | 198.94 (± 0.259) |
| S | 200 | 2 | 0.07 (± 0.047) | 4394.44 (± 990.137) | 167.59 (± 12.887) |
| S | 200 | 3 | 0.03 (± 0.020) | 2643.22 (± 1585.664) | 185.69 (± 14.432) |
| S | 200 | 5 | 0.03 (± 0.021) | 2513.24 (± 1647.668) | 180.92 (± 18.473) |

Table 7.6: Critical set metrics averaged over 5000 local optima of 50 different instances found for six different instance classes.

## 7.4 Summary

In this chapter we dealt with the analysis of search landscapes of instances of the JSOCMSR problem. To this end, different experiments have been conducted to obtain quantitative measures describing the induced landscapes.

In the first section, the ruggedness of landscapes has been investigated. We showed that the destructiveness of a move in a solution, both due to the nature of the move itself or even the spatial scope where structural changes occur have an impact on the change to be expected in the objective value of neighboring solutions. In the subsequent sections we then turned the focus on characteristics of optima in the search space. We started with an analysis of local optima and showed clear differences with respect to solution quality and depth of optima obtained by different neighborhood structures. Some neighborhood structures turned out to yield significantly better local optima, while producing deeper walks through the search space. Other neighborhood structures, in particular those of cardinality $\mathcal{O}(n)$ showed only poor performance with respect to solution quality, suggesting being not highly appropriate for pure intensification phases in the JSOCMSR and may only be used in combinations with others.

In Section 7.3 we finally turned our focus on global optima of instances of moderate size. We found that instances among the studied classes tend to have a vast number of distinct global optima evenly distributed in the search space and showed that characteristics like resource imbalance and the relative utilization of the common resource 0 are among the factors affecting the hardness of an instance. We further started to analyze those optima with respect to potential structural backbones and found that globally optimal

solutions, at least for the studied instances, tend to share characteristics with respect to the critical set. Although some characteristics of jobs constituting these structural backbones may be apparent, like extraordinarily resource demands or highly imbalanced secondary resources, a more thorough analysis is left for future work.

CHAPTER 8

# Computational Results

In this chapter results of a computational study of the devised GVNS for the JSOCMSR will be presented. In the first sections, the most successful variations of our GVNS are studied with respect to different aspects concerning solution quality and temporal behavior, concluded by a comparison to the baseline algorithms presented by Horn et al. [HRB19]. Unless stated otherwise, in all experiments presented in this chapter, ten runs for each of the 50 different instances for each instance class were conducted, with a wall clock time of 900 s per run.

## 8.1 GVNS with Randomized Shaking

In the first experiment we study a GVNS that starts from randomized initial solutions, employs a sequential VND with neighborhoods $\mathcal{N}^I$ as described in Table 8.1.

The set of shaking neighborhood structures $\mathcal{N}^S$ consists of three inversion shaking neighborhood structures and two exponentially increasing sets of shaking neighborhood structures $\mathcal{N}^S_{\mathcal{I},k}$ and $\mathcal{N}^S_{\mathcal{X},k}$, with $k_{\max} = 10$ and a maximum number of shakes of 32, as described in Table 8.2. The first set of intensification neighborhood structures was analytically determined based on results of the performance and landscape analysis of the previous chapters. Furthermore, automatic parameter tuning with the *irace* [LIDLP$^+$16] framework confirmed the selection. Parameter tuning was conducted on a newly generated set of medium sized instances, both balanced and skewed, with $n = 50$, $n = 100$ and $n = 200$. Overall, a training budget, i.e. number of executions of 30000 was configured with a wall-clock time of only 30 s per run for feasibility reasons. The parameter space was designed in a way to tune the number and type of neighborhood structures in $\mathcal{N}^I$ as well as the step function, i.e. *first-fit* and *best-fit*.

Tables 8.3 and 8.4 show the average optimality gap, the number of globally optimal solutions, the average number of iterations as well as the average execution time until termination for balanced and skewed instances respectively. Global optimality of a solution was verified with the tightest lower bounds obtained by Horn et al. [HRB19].

Going in accordance with findings of Chapter 7, balanced instances with $m = 2$ and $m = 5$ turned out to be relatively easy, some almost trivially to solve. For $m = 2$, even for large instances a high percentage of runs starting from completely randomized initial solutions obtained a globally optimal solution way before the time limit was reached.

Even more, the relatively low number of iterations in the GVNS, i.e. shaking followed by a VND, indicates that the instances consist of a vast number of globally optimal solutions evenly distributed in the search space that are reachable with a proper VND from almost any initial solution.

Balanced instances, as shown in Table 8.3, with three secondary resources turned out to be significantly harder to solve for the GVNS. On average, an optimality gap $< 0.3\%$ could be obtained for instances up to $n = 1000$. For $n = 2000$, however, the average optimality gap increases significantly, while the number of iterations drops to 1, indicating a long running VND not being able to terminate in a locally optimal solution.

On the other hand, Table 8.4 shows a quite different behavior for skewed instances. Instances with two secondary resources showed to be more difficult than the corresponding balanced instances. Although for instances up to 1000 jobs an optimality gap of $\leq 0.185\%$ could be obtained, proven globally optimal solutions, i.e. a makespan matching the lower bound from [HRB19], were rarely found. For $m = 3$, the skewed resource utilization and probably more importantly the difference in the utilization of resource 0, showed to be beneficial for the search and allowed to obtain an average optimality gap $\leq 0.035\%$ for $n \leq 1000$. Again, with an increasing number of jobs, the number of iterations drops significantly and thus also affecting the objective in a negative way, resulting in an optimality gap of 0.877 for $n = 2000$.

Figure 8.1 illustrates how the optimality gap evolves over time. The optimality gap is averaged over all 500 executions and was sampled 250 times per second. In addition, $+$ markers indicate the termination of the very first VND to indicate when the search has reached the first valley and enters a more strategic exploration phase. Markers denoted by $\times$, on the other hand, indicate the *average* point of convergence of the search, i.e. when either the optimal solution terminated the search prematurely or no improvement could be obtained anymore.

Generally, for small and medium sized instances the first VND in the search usually manages to obtain high quality solutions (optimality gap close to 1%) within the first 100 s of the search. Furthermore, the gradient during the first VND shows to be rather steep, indicating significant improvement in a relatively small period. Afterwards, once the first optimum is reached, the search enters a *saturation phase*, where the gradient decreases and shaking has to be applied to find improvements.

| | |
|---|---|
| $\mathcal{N}_1^I$ | $\mathcal{N}_\mathcal{B}$ (B) |
| $\mathcal{N}_2^I$ | $\mathcal{N}_\mathcal{X}$ (F) |
| $\mathcal{N}_3^I$ | $\mathcal{N}_\mathcal{I}$ (B) |

Table 8.1: Neighborhood structures $\mathcal{N}^I$ for the VND of the GVNS with purely randomized shaking.

For small and medium sized instances, the GVNS shows to convergence rather fast. Together with the relatively small optimality gap, this may indicate that solutions already rather close to the optimum were found. For large instances, it is shown that the first VND consumes a majority of the execution time and as indicated in Tables 8.3 and 8.4, induces a rather limited number of iterations and in turn an optimality gap significantly above 1%. Since this scalability problem is likely to be caused by the applied VND structure, its behavior will be investigated subsequently in more detail.

In Figure 8.2 we study the contribution of the individual intensification neighborhood structures in the VND in obtaining improved solutions. The plots show the average number of successful moves in all $50 \cdot 10$ runs per instance class. On the one hand, insertion moves are quite dominant for balanced instances with two and five secondary resources. For instances that showed to be more difficult, on the other hand, insertions turned out to be less effective and thus exchange moves constitute a majority of improvements during the search. Generally, this suggests modifications in the VND phase of the search. In the following, we study two different approaches: i) consider alternative progression schemes, particularly a piped VND, or ii) other combinations of neighborhood structures.

Finally, Figure 8.3 shows how the number of successful moves of the respective intensification neighborhood structures evolves in different phases of the search. Note that due to the high variability in execution time of different runs, where some terminated soon, while others reach the time limit, a relative measure of the search progress has been chosen.

Generally, Figure 8.3 suggests that the GVNS starts with a relatively short phase, where moves based on $\mathcal{N}_B$ significantly improve the objective value, until some sort of saturation sets in. Subsequently, the dominance of $\mathcal{N}_\mathcal{X}$ intensifies, while the success rate of insertion-based moves continuously seems to decline. Considering tendencies with respect to the hardness of particular instance classes, for example based on the results of [HRB19], it can be clearly seen that on average $\mathcal{N}_\mathcal{X}$ dominates in hard instances, while only in rather simple instances insertion-based neighborhood structures suffice.

| | | | |
|---|---|---|---|
| $\mathcal{N}_1^S$ | $\mathcal{N}_{I,1}^S$ | $\mathcal{N}_{13}^S$ | $\mathcal{N}_{I,7}^S$ |
| $\mathcal{N}_2^S$ | $\mathcal{N}_{X,1}^S$ | $\mathcal{N}_{14}^S$ | $\mathcal{N}_{X,7}^S$ |
| $\mathcal{N}_3^S$ | $\mathcal{N}_{I,1}^S$ | $\mathcal{N}_{15}^S$ | $\mathcal{N}_{I,10}^S$ |
| $\mathcal{N}_4^S$ | $\mathcal{N}_{X,1}^S$ | $\mathcal{N}_{16}^S$ | $\mathcal{N}_{X,10}^S$ |
| $\mathcal{N}_5^S$ | $\mathcal{N}_{INV-5,1}^S$ | $\mathcal{N}_{17}^S$ | $\mathcal{N}_{I,15}^S$ |
| $\mathcal{N}_6^S$ | $\mathcal{N}_{I,2}^S$ | $\mathcal{N}_{18}^S$ | $\mathcal{N}_{X,15}^S$ |
| $\mathcal{N}_7^S$ | $\mathcal{N}_{X,2}^S$ | $\mathcal{N}_{19}^S$ | $\mathcal{N}_{INV-5,4}^S$ |
| $\mathcal{N}_8^S$ | $\mathcal{N}_{I,3}^S$ | $\mathcal{N}_{20}^S$ | $\mathcal{N}_{I,22}^S$ |
| $\mathcal{N}_9^S$ | $\mathcal{N}_{X,3}^S$ | $\mathcal{N}_{21}^S$ | $\mathcal{N}_{X,22}^S$ |
| $\mathcal{N}_{10}^S$ | $\mathcal{N}_{INV-5,2}^S$ | $\mathcal{N}_{22}^S$ | $\mathcal{N}_{I,32}^S$ |
| $\mathcal{N}_{11}^S$ | $\mathcal{N}_{I,5}^S$ | $\mathcal{N}_{23}^S$ | $\mathcal{N}_{X,32}^S$ |
| $\mathcal{N}_{12}^S$ | $\mathcal{N}_{X,5}^S$ | | |

Table 8.2: Neighborhood structures $\mathcal{N}^S$ for the shaking phase of the GVNS with purely randomized shaking.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] |
|---|---|---|---|---|---|---|---|
| B | 50 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.47) | <0.1 (± <0.01) |
| B | 100 | 2 | 0.000 | <0.01 | 100.0 | 1.2 (± 1.03) | <0.1 (± <0.01) |
| B | 200 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 1.03) | <0.1 (± 0.25) |
| B | 500 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.26) | 5.2 (± 3.80) |
| B | 1000 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.65) | 56.8 (± 46.55) |
| B | 2000 | 2 | 0.032 | 0.11 | 87.4 | 1.0 (± <0.01) | 512.7 (± 242.90) |
| B | 50 | 3 | 0.042 | 0.19 | 91.4 | 7772.3 (± 24251.34) | 84.0 (± 255.92) |
| B | 100 | 3 | 0.109 | 0.28 | 80.2 | 2525.3 (± 5075.09) | 191.4 (± 360.60) |
| B | 200 | 3 | 0.189 | 0.47 | 72.8 | 508.1 (± 801.45) | 271.2 (± 398.77) |
| B | 500 | 3 | 0.267 | 0.43 | 47.2 | 93.7 (± 84.45) | 511.9 (± 423.84) |
| B | 1000 | 3 | 0.218 | 0.34 | 30.4 | 29.0 (± 18.31) | 711.6 (± 296.30) |
| B | 2000 | 3 | 5.542 | 1.71 | 0.2 | 1.0 (± <0.01) | 899.5 (± 10.82) |
| B | 50 | 5 | 0.000 | <0.01 | 100.0 | 6601.4 (± 27227.02) | 55.2 (± 214.02) |
| B | 100 | 5 | 0.000 | <0.01 | 100.0 | 2517.3 (± 10942.19) | 56.4 (± 213.81) |
| B | 200 | 5 | 0.000 | <0.01 | 100.0 | 335.5 (± 1426.67) | 42.6 (± 175.65) |
| B | 500 | 5 | <0.001 | <0.01 | 98.6 | 87.9 (± 140.44) | 82.9 (± 139.66) |
| B | 1000 | 5 | <0.001 | <0.01 | 87.8 | 74.9 (± 77.67) | 321.6 (± 281.11) |
| B | 2000 | 5 | <0.001 | <0.01 | 46.6 | 21.3 (± 21.03) | 768.2 (± 174.02) |

Table 8.3: Performance metrics of a GVNS with randomized shaking for balanced instances.

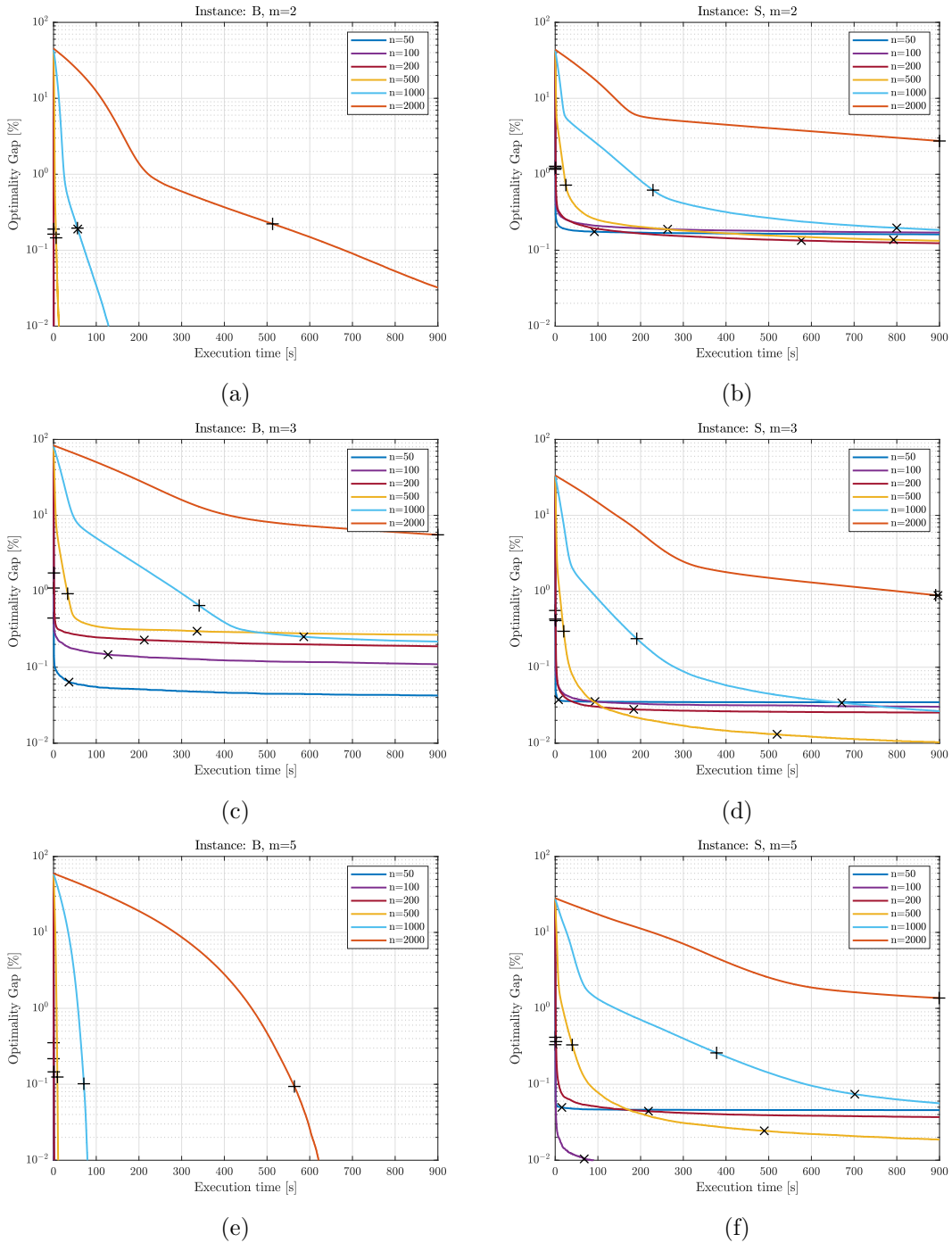
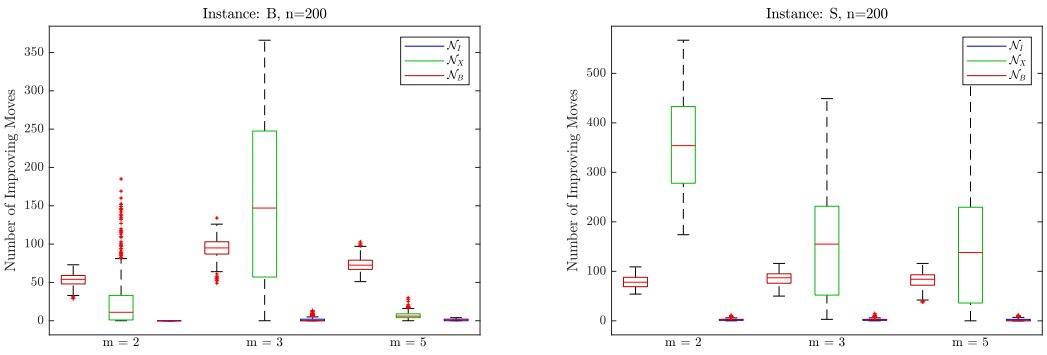



Figure 8.1: Optimality gap over time for the GVNS with randomized shaking. The optimality gap was averaged over the currently best solution sampled 250 times per second. The termination of the first intensification phase is denoted by + markers, while × markers highlight the average point of convergence of the search.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Durations |
|---|---|---|---|---|---|---|---|
| S | 50 | 2 | 0.162 | 0.23 | 42.0 | 87167.9 (± 86973.78) | 524.0 (± 442.43) |
| S | 100 | 2 | 0.171 | 0.32 | 33.2 | 16120.6 (± 12884.92) | 616.8 (± 407.52) |
| S | 200 | 2 | 0.124 | 0.19 | 12.0 | 3865.7 (± 1785.04) | 829.0 (± 209.69) |
| S | 500 | 2 | 0.133 | 0.10 | 0.0 | 566.9 (± 167.56) | 900.0 (± <0.01) |
| S | 1000 | 2 | 0.185 | 0.09 | 0.0 | 199.5 (± 57.17) | 900.0 (± <0.01) |
| S | 2000 | 2 | 2.743 | 0.69 | 0.0 | 1.1 (± 1.20) | 900.0 (± <0.01) |
| S | 50 | 3 | 0.035 | 0.15 | 78.2 | 28641.6 (± 63768.34) | 198.0 (± 371.68) |
| S | 100 | 3 | 0.030 | 0.10 | 82.6 | 4007.0 (± 7881.89) | 189.1 (± 343.13) |
| S | 200 | 3 | 0.025 | 0.11 | 75.0 | 1314.7 (± 1594.43) | 319.7 (± 373.60) |
| S | 500 | 3 | 0.010 | 0.03 | 23.0 | 434.4 (± 216.50) | 754.1 (± 289.18) |
| S | 1000 | 3 | 0.026 | 0.03 | 4.4 | 141.1 (± 49.55) | 879.1 (± 111.66) |
| S | 2000 | 3 | 0.877 | 0.56 | 0.0 | 2.5 (± 8.94) | 900.0 (± <0.01) |
| S | 50 | 5 | 0.046 | 0.14 | 83.7 | 12152.3 (± 30699.34) | 147.1 (± 332.92) |
| S | 100 | 5 | 0.006 | 0.02 | 86.6 | 2102.0 (± 5549.66) | 151.8 (± 316.25) |
| S | 200 | 5 | 0.037 | 0.14 | 68.0 | 774.3 (± 990.80) | 352.8 (± 404.84) |
| S | 500 | 5 | 0.019 | 0.03 | 26.4 | 250.5 (± 139.62) | 725.2 (± 319.60) |
| S | 1000 | 5 | 0.056 | 0.06 | 5.0 | 67.2 (± 39.46) | 875.1 (± 117.95) |
| S | 2000 | 5 | 1.365 | 0.69 | 0.0 | 1.1 (± 1.59) | 900.0 (± 0.18) |

Table 8.4: Performance metrics of a GVNS with randomized shaking for skewed instances.




(a) (b)

Figure 8.2: Distribution of the average number of successful moves of different VND neighborhood structures.

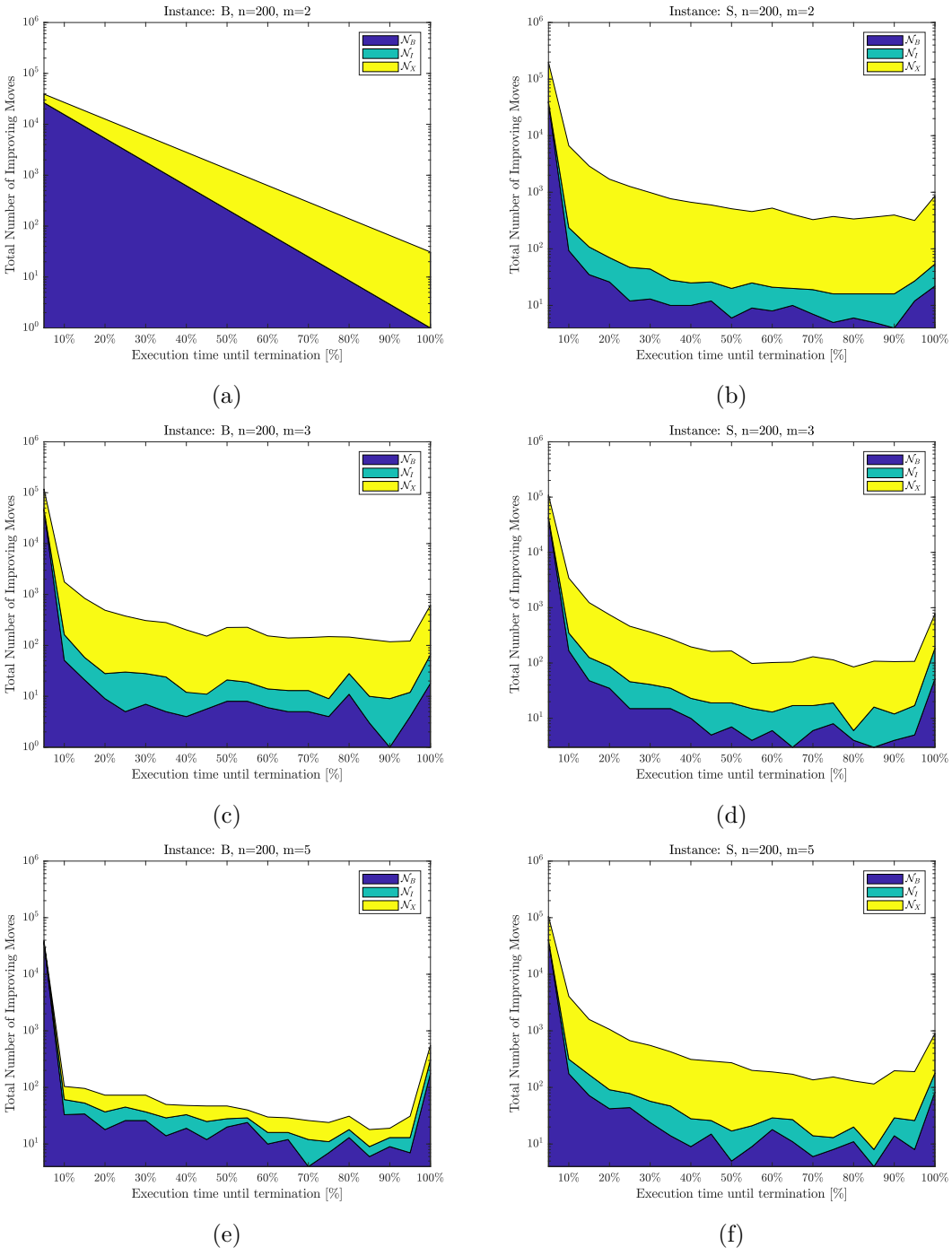



Figure 8.3: Total number of improving move operations in the VND over time. The execution time is given as percentage until termination, either due to solution optimality or exceedance of the time limit. The logarithmic scale on the ordinate is required due to excessively varying scales.

## 8.2 GVNS with Piped VND

The previous section showed a significantly dropping success rate of insertion moves in the VND, once a certain phase in the very beginning of the search is reached. To this end, we now study the impact of a piped VND scheme [DGS06] that sticks to a neighborhood structure, once an improved solution was obtained. Tables 8.5 and 8.6 show performance metrics of the experiment, in addition to p-values of pairwise *Wilcoxcon Rank Sum* tests with $\alpha = 0.01$ to determine statistical significance.

Generally, for small and medium sized instances, no significant improvement with respect to the solution quality could be obtained. However, in Figure 8.4 the temporal behavior of sequential and piped VND for two particularly hard instance classes is illustrated. For instances up to $n = 200$ jobs, the algorithm incorporating the piped VND can obtain solutions with an optimality gap $< 1\%$ already within the first second and also seems to be slightly faster in converging towards the respective high-quality local optimum. For $n = 500$ this becomes more apparent, where a significant improvement can be observed within the first 10 s.

For large instances, a quite similar behavior can be observed. Figure 8.5 shows a generally faster termination of the first VND, yielding an objective value already below $< 1\%$ after approximately 50% of the execution time. Although the number of iterations increases only marginally, as can be seen once more in Tables 8.5 and 8.6, the optimality gap drops below 0.4% even for instances up to $n = 2000$.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| B | 50 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 1.04) | <0.1 (± <0.01) | - |
| B | 100 | 2 | 0.000 | <0.01 | 100.0 | 1.2 (± 1.00) | <0.1 (± <0.01) | - |
| B | 200 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.97) | <0.1 (± 0.06) | - |
| B | 500 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.30) | 2.8 (± 0.70) | - |
| B | 1000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.38) | 26.0 (± 6.56) | - |
| B | 2000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.24) | 223.1 (± 53.30) | $^{+}$**0.000** |
| B | 50 | 3 | 0.044 | 0.20 | 91.2 | 16203.1 (± 51958.43) | 89.5 (± 262.81) | 0.918 |
| B | 100 | 3 | 0.106 | 0.28 | 81.6 | 5048.8 (± 10602.26) | 182.7 (± 351.19) | 0.613 |
| B | 200 | 3 | 0.180 | 0.46 | 75.4 | 1127.6 (± 1893.43) | 250.0 (± 387.32) | 0.408 |
| B | 500 | 3 | 0.251 | 0.41 | 50.2 | 225.0 (± 221.38) | 475.6 (± 433.03) | 0.364 |
| B | 1000 | 3 | 0.203 | 0.32 | 32.6 | 70.6 (± 48.56) | 647.8 (± 372.38) | 0.292 |
| B | 2000 | 3 | 0.375 | 0.39 | 13.4 | 6.1 (± 12.20) | 858.9 (± 119.65) | $^{+}$**0.000** |
| B | 50 | 5 | 0.000 | <0.01 | 100.0 | 8098.4 (± 31589.79) | 55.9 (± 214.81) | - |
| B | 100 | 5 | 0.000 | <0.01 | 100.0 | 2536.7 (± 10389.30) | 56.4 (± 214.43) | - |
| B | 200 | 5 | 0.000 | <0.01 | 100.0 | 389.4 (± 1792.52) | 40.6 (± 176.33) | - |
| B | 500 | 5 | <0.001 | <0.01 | 99.2 | 77.0 (± 157.31) | 64.1 (± 115.96) | 0.364 |
| B | 1000 | 5 | <0.001 | <0.01 | 89.0 | 74.1 (± 84.42) | 297.0 (± 278.49) | 0.551 |
| B | 2000 | 5 | <0.001 | <0.01 | 40.2 | 24.1 (± 21.83) | 786.1 (± 167.28) | 0.020 |

Table 8.5: Performance metrics of a GVNS with randomized shaking and a piped VND for balanced instances. Due to the lack of any variance in some experiments, the test statistic yielded $NaN$. From now on, this is indicated by $-$. In case of statistical significance, $^{+}$ indicates a statistically significant improvement.

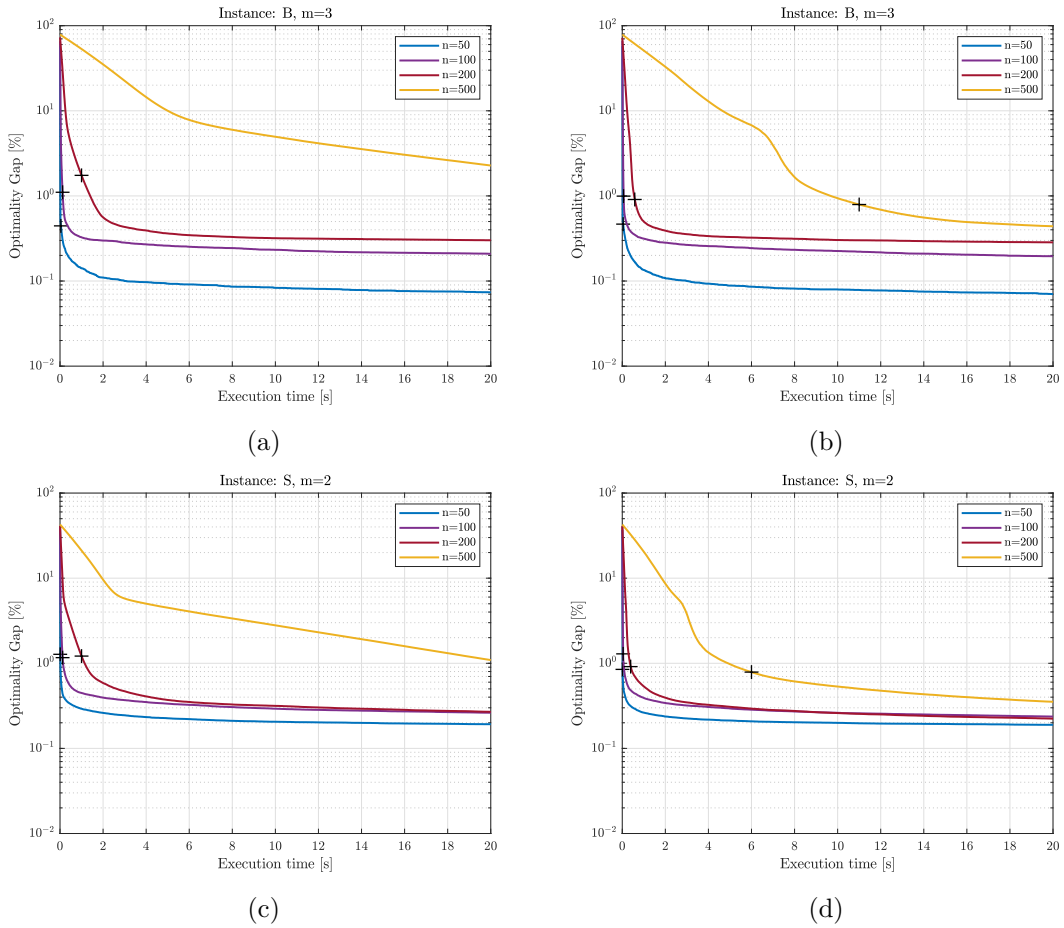



Figure 8.4: Comparison of the optimality gap over time for the GVNS with randomized shaking in Figures 8.4a and 8.4c and the GVNS with a piped VND in Figures 8.4b and 8.4d. The optimality gap was averaged over the currently best solution sampled 250 times per second. The termination of the first intensification phase is denoted by + markers, while × markers highlight the average point of convergence of the search.

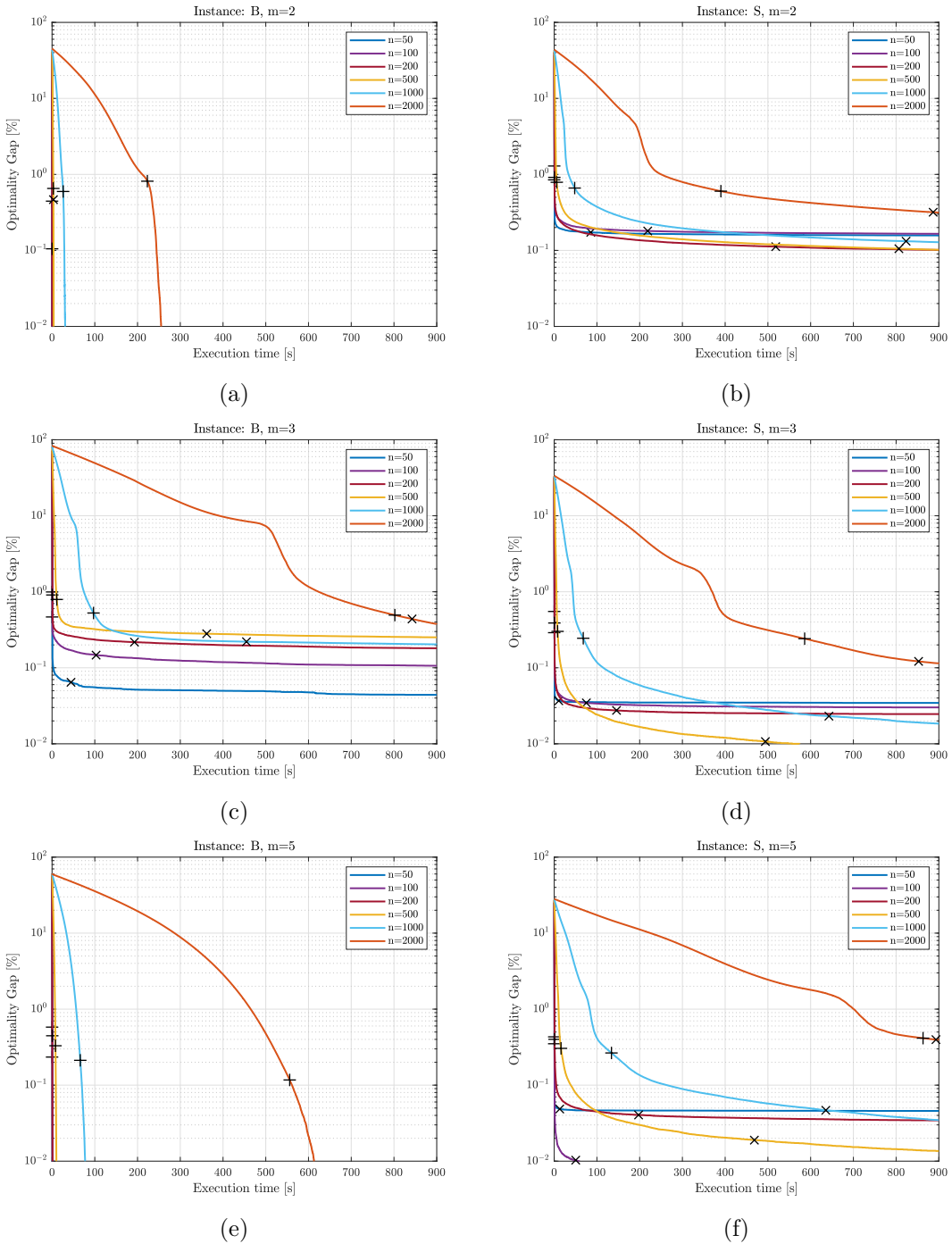



Figure 8.5: Optimality gap over time for the GVNS with randomized shaking and a piped VND scheme. The optimality gap was averaged over the currently best solution sampled 250 times per second. The termination of the first intensification phase is denoted by + markers, while × markers highlight the average point of convergence of the search.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Durations | p-value |
|---|---|---|---|---|---|---|---|---|
| S | 50 | 2 | 0.157 | 0.23 | 42.0 | 163423.5 (± 156814.54) | 524.1 (± 442.60) | 0.910 |
| S | 100 | 2 | 0.165 | 0.31 | 34.4 | 36176.7 (± 29038.21) | 605.8 (± 412.75) | 0.696 |
| S | 200 | 2 | 0.101 | 0.17 | 14.2 | 9762.6 (± 4545.15) | 808.8 (± 238.91) | 0.021 |
| S | 500 | 2 | 0.102 | 0.08 | 0.2 | 1555.0 (± 492.19) | 900.0 (± 0.09) | $^{+}$**0.000** |
| S | 1000 | 2 | 0.128 | 0.06 | 0.0 | 495.1 (± 125.98) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 2000 | 2 | 0.315 | 0.12 | 0.0 | 118.3 (± 60.00) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 50 | 3 | 0.034 | 0.15 | 82.2 | 46519.6 (± 96172.75) | 197.9 (± 372.10) | 0.997 |
| S | 100 | 3 | 0.030 | 0.10 | 82.0 | 6893.4 (± 13818.26) | 180.8 (± 344.00) | 0.834 |
| S | 200 | 3 | 0.025 | 0.11 | 76.8 | 2158.5 (± 3008.48) | 272.4 (± 370.46) | 0.513 |
| S | 500 | 3 | 0.008 | 0.02 | 25.2 | 842.4 (± 436.33) | 738.4 (± 302.83) | 0.040 |
| S | 1000 | 3 | 0.018 | 0.02 | 6.8 | 279.6 (± 100.60) | 857.9 (± 165.33) | $^{+}$**0.001** |
| S | 2000 | 3 | 0.114 | 0.11 | 0.4 | 42.8 (± 39.38) | 898.3 (± 29.46) | $^{+}$**0.000** |
| S | 50 | 5 | 0.046 | 0.14 | 83.7 | 20422.5 (± 50752.45) | 147.2 (± 332.89) | 0.995 |
| S | 100 | 5 | 0.006 | 0.02 | 88.4 | 3295.3 (± 8150.13) | 142.0 (± 313.17) | 0.916 |
| S | 200 | 5 | 0.034 | 0.14 | 73.0 | 1300.8 (± 1712.84) | 314.8 (± 385.84) | 0.120 |
| S | 500 | 5 | 0.014 | 0.03 | 30.0 | 428.8 (± 252.69) | 688.3 (± 348.19) | 0.023 |
| S | 1000 | 5 | 0.035 | 0.04 | 6.4 | 160.4 (± 63.00) | 868.6 (± 133.87) | $^{+}$**0.000** |
| S | 2000 | 5 | 0.393 | 0.26 | 0.0 | 5.3 (± 14.24) | 900.1 (± 0.29) | $^{+}$**0.000** |

Table 8.6: Performance metrics of a GVNS with randomized shaking and a piped VND for skewed instances.

## 8.3 GVNS with Intensified Shaking

In this section, we study the behavior of intensified shaking in our GVNS. To this end, the move based shaking neighborhoods $\mathcal{N}^S_{I,i}$,$\mathcal{N}^S_{X,i}$ and $\mathcal{N}^S_{INV-5,i}$ shown in Table 8.2 are replaced by the corresponding intensifying shaking neighborhood structures. In addition, as last shaking neighborhood structure the CP-based repair and destroy neighborhood $\mathcal{N}^S_{CP}$ with $k = 15$ was incorporated. Table 8.7 lists the employed shaking neighborhood structures. Tables 8.8 and 8.9 again show several performance metrics averaged over all runs and p-values of pairwise *Wilcoxcon Rank Sum* tests to determine statistical significance against the GVNS shown in Section 8.2.

In general, it turned out that purely randomized shaking performs quite similar to the intensified shaking approach. However, for some larger instances, a slight but still statistically significant improvement could be obtained. For balanced instances, this only affects large instances with five secondary resources, where the obtained optimality gap is already below 0.001%. For skewed instances, a minor improvement in large instances could be achieved, in particular, for instances with two secondary resources with $n \geq 500$.

Generally, the number of successful shakes in our GVNS is compared to the number of iterations shown in Tables 8.8 and 8.9 relatively low. A comparison of successful shaking invocations of skewed instances is shown in Figure 8.6. On the one hand, since the average optimality gap is still relatively low, this suggests that a large number of poor local optima can already be avoided by the VND. Furthermore, it can be observed that the number of found improvements significantly decreases with the destructiveness introduced by the shaking neighborhood. So while small perturbation tend to guide the search into improving areas relatively often, large perturbations, like for example $\mathcal{N}^S_{22}$

| | | | |
|---|---|---|---|
| $\mathcal{N}^S_1$ | $\mathcal{N}^S_{\text{G}-I,1}$ | $\mathcal{N}^S_{14}$ | $\mathcal{N}^S_{\text{G}-E,7}$ |
| $\mathcal{N}^S_2$ | $\mathcal{N}^S_{\text{G}-E,1}$ | $\mathcal{N}^S_{15}$ | $\mathcal{N}^S_{\text{G}-I,10}$ |
| $\mathcal{N}^S_3$ | $\mathcal{N}^S_{\text{G}-I,1}$ | $\mathcal{N}^S_{16}$ | $\mathcal{N}^S_{\text{G}-E,10}$ |
| $\mathcal{N}^S_4$ | $\mathcal{N}^S_{\text{G}-E,1}$ | $\mathcal{N}^S_{17}$ | $\mathcal{N}^S_{\text{G}-I,15}$ |
| $\mathcal{N}^S_5$ | $\mathcal{N}^S_{\text{G}-INV-5,1}$ | $\mathcal{N}^S_{18}$ | $\mathcal{N}^S_{\text{G}-E,15}$ |
| $\mathcal{N}^S_6$ | $\mathcal{N}^S_{\text{G}-I,2}$ | $\mathcal{N}^S_{19}$ | $\mathcal{N}^S_{\text{G}-INV-5,4}$ |
| $\mathcal{N}^S_7$ | $\mathcal{N}^S_{\text{G}-E,2}$ | $\mathcal{N}^S_{20}$ | $\mathcal{N}^S_{\text{G}-I,22}$ |
| $\mathcal{N}^S_8$ | $\mathcal{N}^S_{\text{G}-I,3}$ | $\mathcal{N}^S_{21}$ | $\mathcal{N}^S_{\text{G}-E,22}$ |
| $\mathcal{N}^S_9$ | $\mathcal{N}^S_{\text{G}-E,3}$ | $\mathcal{N}^S_{22}$ | $\mathcal{N}^S_{\text{G}-I,32}$ |
| $\mathcal{N}^S_{10}$ | $\mathcal{N}^S_{\text{G}-INV-5,2}$ | $\mathcal{N}^S_{23}$ | $\mathcal{N}^S_{\text{G}-E,32}$ |
| $\mathcal{N}^S_{11}$ | $\mathcal{N}^S_{\text{G}-I,5}$ | $\mathcal{N}^S_{24}$ | $\mathcal{N}^S_{\text{G}-INV-5,5}$ |
| $\mathcal{N}^S_{12}$ | $\mathcal{N}^S_{\text{G}-E,5}$ | $\mathcal{N}^S_{25}$ | $\mathcal{N}^S_{CP-15,1}$ |
| $\mathcal{N}^S_{13}$ | $\mathcal{N}^S_{\text{G}-I,7}$ | | |

Table 8.7: Neighborhood structures $\mathcal{N}^S$ for the shaking phase of the GVNS with intensified shaking..

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| B | 50 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.99) | <0.1 (± <0.01) | - |
| B | 100 | 2 | 0.000 | <0.01 | 100.0 | 1.2 (± 1.13) | <0.1 (± <0.01) | - |
| B | 200 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.93) | <0.1 (± 0.09) | - |
| B | 500 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.09) | 2.9 (± 0.74) | - |
| B | 1000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.54) | 26.7 (± 6.20) | - |
| B | 2000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.59) | 226.6 (± 54.34) | - |
| B | 50 | 3 | 0.055 | 0.23 | 90.2 | 1338.6 (± 3770.69) | 95.8 (± 270.83) | 0.534 |
| B | 100 | 3 | 0.115 | 0.30 | 80.4 | 2501.8 (± 4792.58) | 188.0 (± 358.74) | 0.627 |
| B | 200 | 3 | 0.171 | 0.44 | 76.8 | 1265.9 (± 2128.00) | 240.2 (± 380.89) | 0.620 |
| B | 500 | 3 | 0.236 | 0.40 | 50.8 | 266.7 (± 268.15) | 472.6 (± 432.41) | 0.643 |
| B | 1000 | 3 | 0.196 | 0.32 | 33.6 | 75.8 (± 51.83) | 640.7 (± 373.34) | 0.748 |
| B | 2000 | 3 | 0.388 | 0.40 | 13.2 | 6.2 (± 12.44) | 864.0 (± 100.65) | 0.565 |
| B | 50 | 5 | 0.000 | <0.01 | 100.0 | 2262.9 (± 17136.12) | 59.3 (± 217.52) | - |
| B | 100 | 5 | 0.000 | <0.01 | 100.0 | 2233.3 (± 8717.02) | 59.1 (± 215.72) | - |
| B | 200 | 5 | 0.000 | <0.01 | 100.0 | 655.5 (± 2926.67) | 41.1 (± 175.92) | - |
| B | 500 | 5 | <0.001 | <0.01 | 98.2 | 156.1 (± 363.43) | 75.2 (± 150.25) | 0.162 |
| B | 1000 | 5 | <0.001 | <0.01 | 87.6 | 119.9 (± 143.24) | 298.2 (± 281.62) | 0.501 |
| B | 2000 | 5 | <0.001 | <0.01 | 48.0 | 29.2 (± 27.68) | 759.5 (± 181.16) | $^{+}$**0.002** |

Table 8.8: Performance metrics of a GVNS with intensified randomized shaking and a piped VND for balanced instances.

in Figure 8.6a that conducts 32 random insertions, rarely lead to improved solutions. So once the search reaches the first local optima, it manages to find a majority of improving regions already in close surrounding areas, while large perturbations, as it would be the case for example in random restart, approaches rarely lead to significant improvements. More intensifying perturbations, like the *destroy & repair* based shaking neighborhood $\mathcal{N}^S_{25}$ in Figures 8.6b and 8.6d shows to be advantageous for the shown instance class. However, for other instance classes this behavior could not be observed.

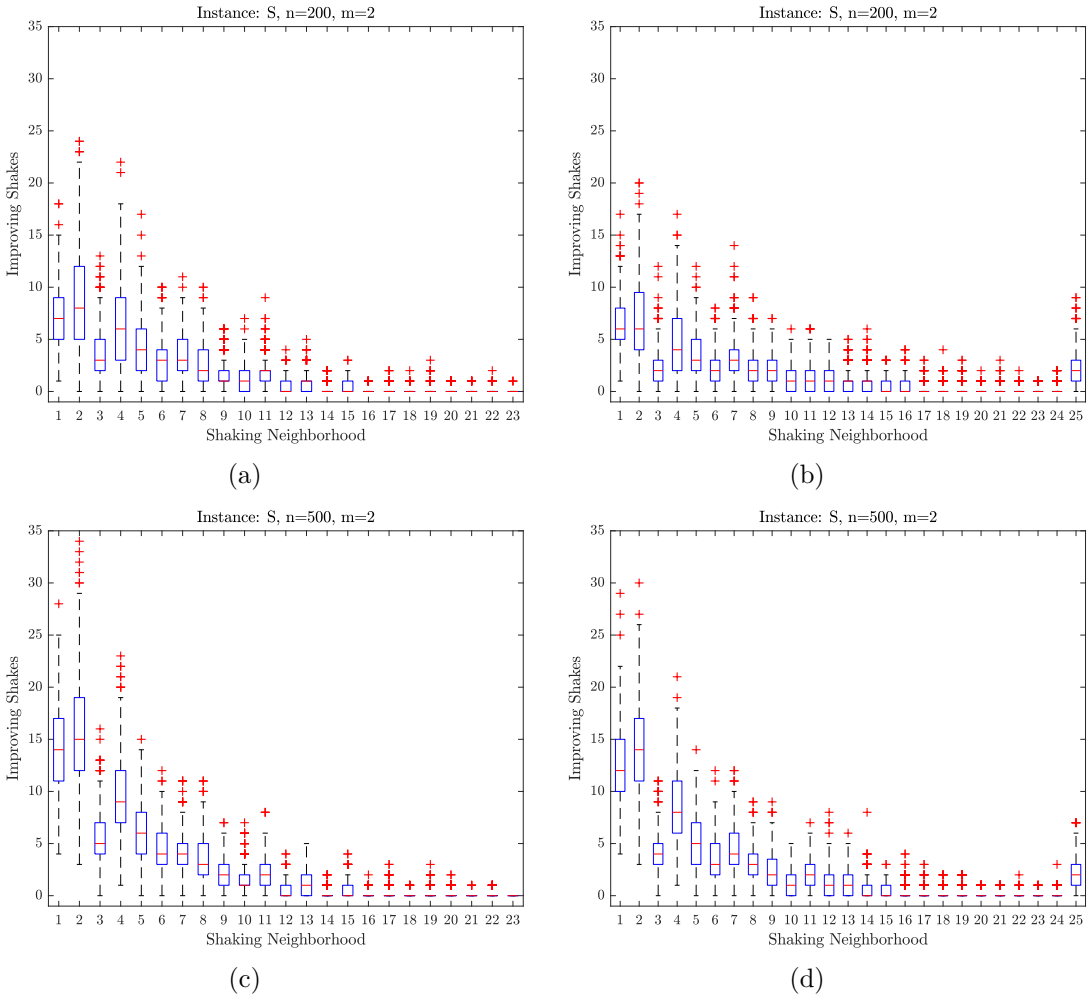



Figure 8.6: Comparison of the success rate of shaking neighborhood structures in purely randomized shaking in Figures 8.6a and 8.6c and intensified shaking in Figures 8.6b and 8.6d.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| S | 50 | 2 | 0.181 | 0.25 | 41.6 | 7140.9 (± 5733.58) | 540.8 (± 432.49) | 0.337 |
| S | 100 | 2 | 0.180 | 0.32 | 32.4 | 8204.2 (± 5227.83) | 631.2 (± 397.27) | 0.310 |
| S | 200 | 2 | 0.099 | 0.16 | 12.4 | 8397.1 (± 2756.03) | 818.8 (± 233.29) | 0.844 |
| S | 500 | 2 | 0.083 | 0.07 | 0.2 | 1872.8 (± 518.43) | 899.9 (± 1.66) | $^{+}$**0.002** |
| S | 1000 | 2 | 0.100 | 0.05 | 0.0 | 530.6 (± 146.36) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 2000 | 2 | 0.225 | 0.09 | 0.0 | 118.9 (± 59.20) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 50 | 3 | 0.035 | 0.15 | 80.8 | 2713.3 (± 4934.10) | 212.4 (± 371.90) | 0.600 |
| S | 100 | 3 | 0.031 | 0.10 | 81.4 | 3176.1 (± 5218.62) | 210.7 (± 348.82) | 0.774 |
| S | 200 | 3 | 0.025 | 0.11 | 76.4 | 2580.5 (± 3374.78) | 282.3 (± 371.21) | 0.961 |
| S | 500 | 3 | 0.007 | 0.02 | 30.8 | 1131.6 (± 653.64) | 688.3 (± 341.13) | 0.024 |
| S | 1000 | 3 | 0.021 | 0.03 | 9.4 | 374.7 (± 150.17) | 843.1 (± 190.45) | 0.700 |
| S | 2000 | 3 | 0.114 | 0.11 | 1.0 | 48.9 (± 46.54) | 898.4 (± 20.00) | 0.695 |
| S | 50 | 5 | 0.047 | 0.14 | 83.7 | 6024.8 (± 29114.87) | 150.0 (± 332.51) | 0.978 |
| S | 100 | 5 | 0.006 | 0.02 | 88.4 | 3000.9 (± 8113.79) | 142.4 (± 313.60) | 0.987 |
| S | 200 | 5 | 0.034 | 0.14 | 74.8 | 1622.5 (± 2180.32) | 292.8 (± 378.09) | 0.556 |
| S | 500 | 5 | 0.014 | 0.03 | 33.8 | 599.5 (± 372.55) | 665.0 (± 357.12) | 0.251 |
| S | 1000 | 5 | 0.042 | 0.05 | 10.4 | 197.3 (± 87.12) | 839.4 (± 192.04) | 0.531 |
| S | 2000 | 5 | 0.401 | 0.27 | 0.0 | 6.4 (± 18.09) | 900.1 (± 0.23) | 0.708 |

Table 8.9: Performance metrics of a GVNS with intensified randomized shaking and a piped VND for skewed instances.

## 8.4 GVNS and LLBH

In this section, we study the impact of starting the GVNS with a piped VND scheme, as presented in Section 8.2, from initial solution obtained with a proper construction heuristic. As stated previously, in this work we made use of the LLBH devised by Horn et al. [HRB19]. In the following experiments, obtaining the initial solution with LLBH was part of the maximum execution time of 900 s. The results are shown in Tables 8.10 and 8.11.

Generally, our experiments confirm the results shown by Horn et al. regarding balanced instances with $m = 2$, where any instance could be solved to proven optimality in only a fraction of the maximum execution time. For balanced instances with $m = 3$, particularly large instances with $n \geq 500$, a statistically significant improvement was obtained. Even more, the percentage of obtained global optima increased from 13% for $n = 2000$, to 76.4%, together with a significant reduction in the average execution time that drops from $\approx 859$ s to 228 s. A very similar behavior can be observed for $m = 5$, where also both the number of found global optima and the average execution times improve significantly, although the difference in average optimality gap is $< 0.0005\%$.

For skewed instances, on the other hand, an improvement in the optimality gap could only be obtained for large instances with $m = 2$. Regarding execution times or found optima no significant difference could be observed. However, for $m = 3$ and $m = 5$, starting from the solution obtained with LLBH turns out to be disadvantageous and goes in accordance with the findings of Horn et al. who showed a comparably poor performance of the LLBH for skewed instances.

But more interestingly is probably the temporal behavior as shown in Figure 8.7. On the one hand, for balanced and skewed instances with $m = 3$ and $m = 2$ respectively, it

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| B | 50 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± <0.01) | <0.1 (± <0.01) | - |
| B | 100 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± <0.01) | <0.1 (± <0.01) | - |
| B | 200 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± <0.01) | <0.1 (± <0.01) | - |
| B | 500 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± <0.01) | <0.1 (± <0.01) | - |
| B | 1000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± <0.01) | 2.1 (± 0.53) | - |
| B | 2000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± <0.01) | 16.4 (± 3.56) | - |
| B | 50 | 3 | 0.043 | 0.19 | 91.4 | 14213.9 (± 45075.61) | 83.0 (± 255.52) | 0.927 |
| B | 100 | 3 | 0.100 | 0.26 | 80.6 | 4885.4 (± 9947.16) | 183.7 (± 358.13) | 0.788 |
| B | 200 | 3 | 0.150 | 0.40 | 79.2 | 882.0 (± 1701.69) | 200.4 (± 365.80) | 0.160 |
| B | 500 | 3 | 0.117 | 0.24 | 64.2 | 132.2 (± 186.54) | 328.8 (± 430.56) | $^{+}$**0.000** |
| B | 1000 | 3 | 0.059 | 0.17 | 74.6 | 22.4 (± 38.17) | 233.9 (± 391.23) | $^{+}$**0.000** |
| B | 2000 | 3 | 0.031 | 0.10 | 76.4 | 7.3 (± 11.49) | 228.2 (± 376.04) | $^{+}$**0.000** |
| B | 50 | 5 | 0.000 | <0.01 | 100.0 | 8469.5 (± 34896.66) | 55.3 (± 213.93) | - |
| B | 100 | 5 | 0.000 | <0.01 | 100.0 | 2154.2 (± 8726.40) | 55.8 (± 214.57) | - |
| B | 200 | 5 | 0.000 | <0.01 | 100.0 | 261.0 (± 1268.88) | 36.2 (± 176.52) | - |
| B | 500 | 5 | 0.000 | <0.01 | 100.0 | 2.3 (± 8.71) | 1.2 (± 8.34) | 0.045 |
| B | 1000 | 5 | 0.000 | <0.01 | 100.0 | 1.8 (± 6.91) | 5.5 (± 31.63) | $^{+}$**0.000** |
| B | 2000 | 5 | <0.001 | <0.01 | 99.4 | 1.3 (± 3.49) | 20.6 (± 70.49) | $^{+}$**0.000** |

Table 8.10: Performance metrics of a GVNS with randomized shaking, a piped VND and starting with initial solutions obtained with LLBH for balanced instances.

can be observed that the search already starts from rather good initial solutions such that the initial intensification phase terminates rather soon. For the balanced instances, the search converges rather fast, suggesting that the initial solution obtained with the construction heuristic is already rather close to an optimum.

For skewed instances with $n = 2000$ and $m = 3$ or $m = 5$, the improved initial solution turns out to be disadvantageous. Even though the search already starts with an optimality gap of $\approx 5\%$, the gradient of improvement is rather moderate, while still requiring a relatively large amount of the available execution time. So although the search continuously manages to explore slightly improving areas, no significant improvements occur, indicating that the search gets stuck in a rather *strong* locally optimal region.

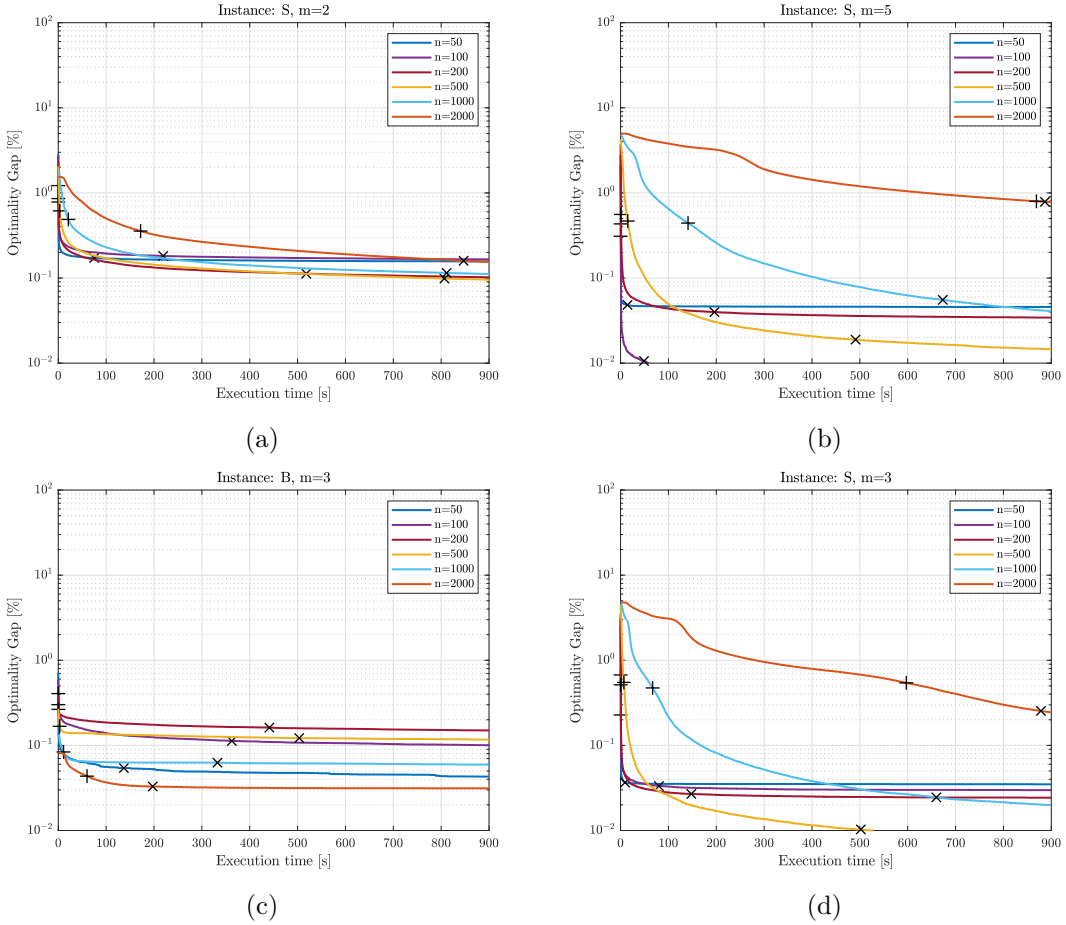



Figure 8.7: Optimality gap over time for the GVNS with a piped VND starting from solutions obtained by LLBH. The optimality gap was averaged over the currently best solution sampled 250 times per second. The termination of the first intensification phase is denoted by + markers, while × markers highlight the average point of convergence of the search.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| S | 50 | 2 | 0.157 | 0.23 | 42.0 | 163986.0 (± 150425.54) | 523.5 (± 443.02) | 0.919 |
| S | 100 | 2 | 0.166 | 0.31 | 34.8 | 36384.9 (± 27445.26) | 605.5 (± 414.14) | 0.993 |
| S | 200 | 2 | 0.102 | 0.17 | 15.2 | 9754.0 (± 4134.16) | 799.9 (± 253.88) | 0.953 |
| S | 500 | 2 | 0.096 | 0.08 | 0.0 | 1581.9 (± 395.92) | 900.0 (± <0.01) | 0.528 |
| S | 1000 | 2 | 0.112 | 0.05 | 0.0 | 482.0 (± 105.01) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 2000 | 2 | 0.154 | 0.05 | 0.0 | 161.5 (± 43.81) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 50 | 3 | 0.035 | 0.15 | 82.2 | 46204.3 (± 93060.36) | 196.8 (± 371.69) | 0.994 |
| S | 100 | 3 | 0.030 | 0.10 | 83.6 | 6641.3 (± 13315.22) | 172.4 (± 334.96) | 0.538 |
| S | 200 | 3 | 0.024 | 0.11 | 76.4 | 2127.0 (± 2973.61) | 270.1 (± 367.53) | 0.957 |
| S | 500 | 3 | 0.008 | 0.02 | 26.4 | 816.1 (± 381.50) | 728.5 (± 310.70) | 0.894 |
| S | 1000 | 3 | 0.020 | 0.02 | 5.8 | 296.8 (± 87.90) | 870.4 (± 135.53) | 0.339 |
| S | 2000 | 3 | 0.244 | 0.18 | 0.0 | 29.2 (± 34.22) | 900.0 (± 0.04) | $^{-}$**0.000** |
| S | 50 | 5 | 0.046 | 0.14 | 83.7 | 19440.6 (± 46152.16) | 147.1 (± 332.92) | 0.996 |
| S | 100 | 5 | 0.006 | 0.02 | 88.6 | 3336.5 (± 8538.66) | 138.6 (± 311.76) | 0.928 |
| S | 200 | 5 | 0.034 | 0.14 | 72.2 | 1224.3 (± 1523.16) | 309.9 (± 385.10) | 0.868 |
| S | 500 | 5 | 0.015 | 0.03 | 26.4 | 444.0 (± 218.59) | 722.3 (± 324.24) | 0.383 |
| S | 1000 | 5 | 0.041 | 0.04 | 2.6 | 167.9 (± 49.06) | 891.0 (± 64.01) | 0.011 |
| S | 2000 | 5 | 0.777 | 0.36 | 0.0 | 2.4 (± 6.49) | 900.1 (± 0.35) | $^{-}$**0.000** |

Table 8.11: Performance metrics of a GVNS with randomized shaking, a piped VND and starting with initial solutions obtained with LLBH for skewed instances.

## 8.5 GVNS with Range Limited Neighborhood VND

In the previous sections it was shown that a majority of improving neighbors in hard instances are obtained by exchange moves. However, intensification phases purely based on exchange moves also turned out as non-effective, due to the increased complexity of the evaluation scheme and the lack of diversity introduced by complementing neighborhood structures.

Thus, in this section the impact of one of the presented neighborhood pruning approaches, based on moves of limited spatial scope will be presented. Although we also performed some tests with other pruning approaches presented in Section 5.4.1, they soon turned out to be not of any significant advantage in the search.

The VND was adapted to incorporate an additional exchange neighborhood structure with a range limit of 50. Based on the performance evaluation in Chapter 6, we claim a limit of 50 provides a reasonable tradeoff between spatial scope of the neighborhood structure and evaluation efficiency. The resulting VND incorporated in the GVNS with purely randomized shaking, as in Section 8.2, is shown in Table 8.12.

Tables 8.13 and 8.14 show the usual set of performance metrics. Again, for particularly large instances, significant improvements could be obtained compared to the GVNS with piped VND baseline. With respect to the shown performance metrics, the algorithms behave in generally quite similarly, although the extended VND scheme induces a lower number of iterations.

Interestingly, for large instances the approach becomes advantageous with respect to the temporal behavior, as shown in Figure 8.8. Particularly in the beginning of the

| | |
|---|---|
| $\mathcal{N}_1^I$ | $\mathcal{N}_{\mathcal{X}-50}$ (B) |
| $\mathcal{N}_2^I$ | $\mathcal{N}_{\mathcal{B}}$ (B) |
| $\mathcal{N}_3^I$ | $\mathcal{N}_{\mathcal{X}}$ (F) |
| $\mathcal{N}_4^I$ | $\mathcal{N}_{\mathcal{I}}$ (F) |

Table 8.12: Neighborhood structures $\mathcal{N}^I$ for the VND of the GVNS with range-limited neighborhood structures.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| B | 50 | 2 | 0.000 | <0.01 | 100.0 | 1.3 (± 1.41) | <0.1 (± 0.19) | - |
| B | 100 | 2 | 0.000 | <0.01 | 100.0 | 1.3 (± 1.50) | <0.1 (± <0.01) | - |
| B | 200 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.56) | <0.1 (± 0.04) | - |
| B | 500 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.19) | 2.0 (± 0.94) | - |
| B | 1000 | 2 | 0.000 | <0.01 | 100.0 | 1.0 (± 0.32) | 13.1 (± 5.37) | - |
| B | 2000 | 2 | 0.000 | <0.01 | 100.0 | 1.1 (± 0.54) | 83.7 (± 35.04) | - |
| B | 50 | 3 | 0.050 | 0.22 | 91.2 | 9049.6 (± 28961.58) | 86.1 (± 257.04) | 0.951 |
| B | 100 | 3 | 0.112 | 0.29 | 79.6 | 3484.7 (± 6956.93) | 192.8 (± 363.75) | 0.468 |
| B | 200 | 3 | 0.176 | 0.45 | 74.0 | 958.9 (± 1546.80) | 257.8 (± 392.59) | 0.708 |
| B | 500 | 3 | 0.260 | 0.42 | 47.0 | 199.2 (± 181.71) | 499.8 (± 433.66) | 0.449 |
| B | 1000 | 3 | 0.216 | 0.33 | 31.0 | 57.6 (± 37.87) | 644.6 (± 386.02) | 0.417 |
| B | 2000 | 3 | 0.288 | 0.34 | 15.0 | 12.2 (± 9.95) | 815.7 (± 212.37) | $^{+}$**0.000** |
| B | 50 | 5 | <0.001 | <0.01 | 99.4 | 7944.4 (± 32899.97) | 60.3 (± 223.63) | 0.083 |
| B | 100 | 5 | 0.000 | <0.01 | 100.0 | 2144.1 (± 8792.59) | 55.6 (± 213.65) | - |
| B | 200 | 5 | 0.000 | <0.01 | 100.0 | 426.2 (± 1894.64) | 40.1 (± 175.89) | - |
| B | 500 | 5 | 0.000 | <0.01 | 100.0 | 84.3 (± 169.53) | 35.6 (± 74.60) | 0.045 |
| B | 1000 | 5 | <0.001 | <0.01 | 96.0 | 93.9 (± 153.21) | 130.9 (± 205.96) | $^{+}$**0.000** |
| B | 2000 | 5 | <0.001 | <0.01 | 86.6 | 61.6 (± 71.61) | 302.7 (± 299.52) | $^{+}$**0.000** |

Table 8.13: GVNS and with a range-limited VND for balanced instances.

search, the alternative VND scheme advances with a steeper descent and ends up in a comparable local optimum faster. From the point of first saturation on, i.e. as soon as the shaking phase is triggered, the temporal behavior becomes quite similarly and a phase of small, but continuously improvements of the objective sets in.

Finally, Figure 8.9 again compares success rates of the different intensification neighborhood structures. Similar to VND presented in the previous sections, exchange moves still dominate the search, but a non-negligible amount can be conducted by the computationally more efficient neighborhood structure $\mathcal{N}_{\mathcal{X}-50}$.

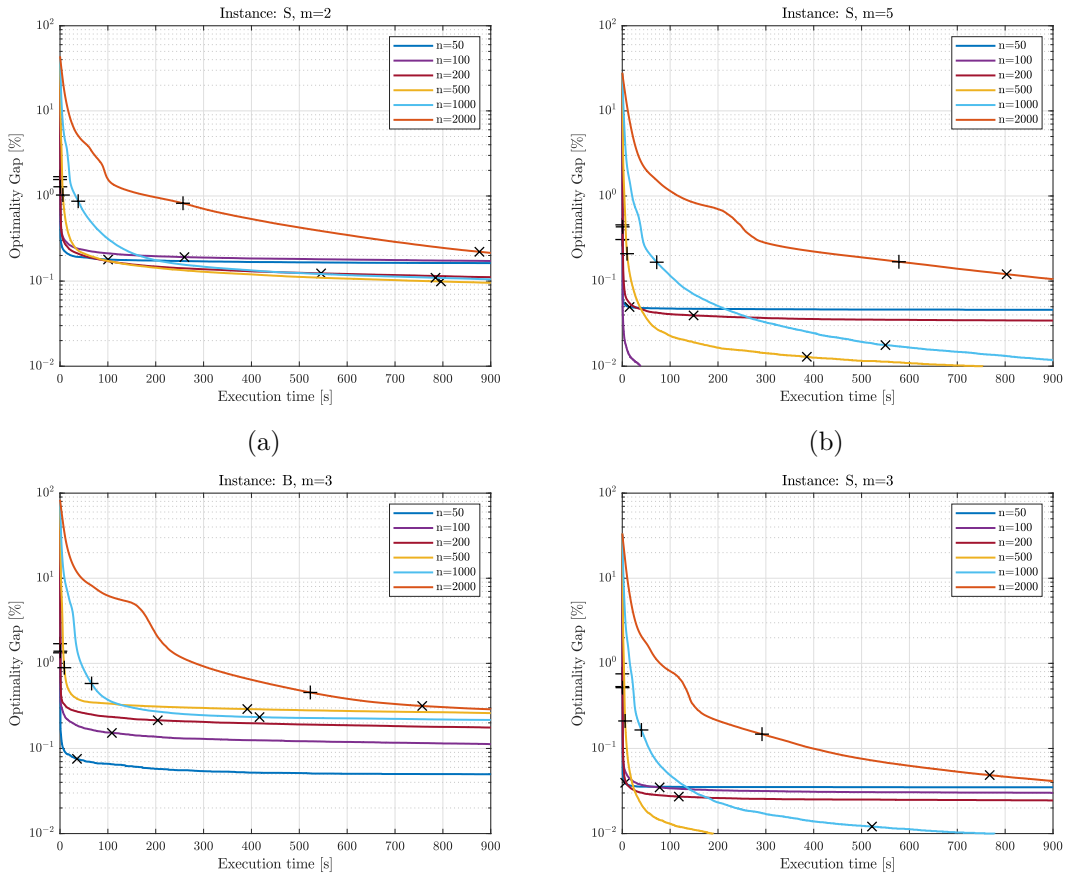



Figure 8.8: Optimality gap over time for the GVNS making use of range limited neighborhoods. The optimality gap was averaged over the currently best solution sampled 250 times per second. The termination of the first intensification phase is denoted by + markers, while × markers highlight the average point of convergence of the search.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| S | 50 | 2 | 0.163 | 0.23 | 42.0 | 76726.9 (± 70026.00) | 527.3 (± 439.69) | 0.879 |
| S | 100 | 2 | 0.172 | 0.32 | 33.8 | 16765.8 (± 12366.67) | 613.0 (± 407.78) | 0.790 |
| S | 200 | 2 | 0.111 | 0.18 | 14.8 | 5312.5 (± 2127.65) | 807.8 (± 241.52) | 0.616 |
| S | 500 | 2 | 0.095 | 0.08 | 0.0 | 923.6 (± 238.93) | 900.0 (± <0.01) | 0.192 |
| S | 1000 | 2 | 0.105 | 0.06 | 0.0 | 265.7 (± 57.93) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 2000 | 2 | 0.214 | 0.11 | 0.0 | 74.0 (± 25.71) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 50 | 3 | 0.035 | 0.15 | 82.0 | 26018.3 (± 51650.06) | 198.6 (± 372.88) | 0.928 |
| S | 100 | 3 | 0.030 | 0.10 | 82.8 | 4182.8 (± 8287.14) | 177.6 (± 338.89) | 0.749 |
| S | 200 | 3 | 0.025 | 0.11 | 78.8 | 1309.9 (± 2005.60) | 240.0 (± 363.03) | 0.493 |
| S | 500 | 3 | 0.006 | 0.02 | 42.4 | 614.0 (± 410.63) | 598.3 (± 376.87) | $^{+}$**0.000** |
| S | 1000 | 3 | 0.009 | 0.02 | 19.2 | 232.2 (± 108.98) | 776.0 (± 273.47) | $^{+}$**0.000** |
| S | 2000 | 3 | 0.041 | 0.05 | 5.8 | 73.9 (± 33.95) | 877.4 (± 103.40) | $^{+}$**0.000** |
| S | 50 | 5 | 0.046 | 0.14 | 83.7 | 14704.2 (± 35292.61) | 147.1 (± 332.92) | 0.998 |
| S | 100 | 5 | 0.006 | 0.02 | 88.4 | 2532.7 (± 6383.77) | 139.2 (± 311.50) | 0.987 |
| S | 200 | 5 | 0.034 | 0.14 | 77.2 | 846.0 (± 1245.26) | 252.1 (± 374.22) | 0.179 |
| S | 500 | 5 | 0.009 | 0.02 | 46.8 | 308.7 (± 226.60) | 555.3 (± 403.47) | $^{+}$**0.000** |
| S | 1000 | 5 | 0.012 | 0.02 | 22.2 | 144.3 (± 64.08) | 765.6 (± 272.99) | $^{+}$**0.000** |
| S | 2000 | 5 | 0.105 | 0.10 | 2.0 | 23.4 (± 23.44) | 892.8 (± 60.43) | $^{+}$**0.000** |

Table 8.14: GVNS and with a range-limited VND for skewed instances.

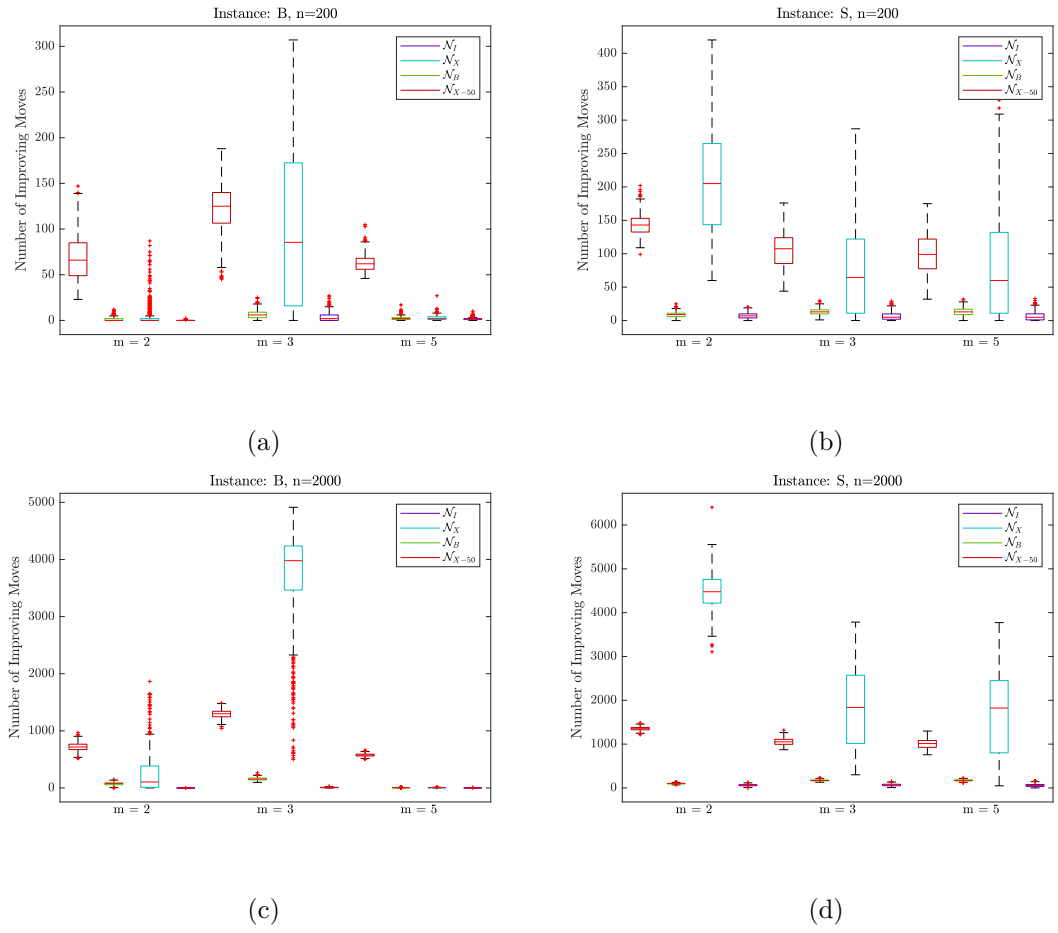



Figure 8.9: Distribution of successful moves of different VND neighborhood structures of the GVNS employing range limited neighborhood structures in the VND.

## 8.6 Neighborhood Pruning based on the Critical Set

Although the experiments of the previous sections demonstrated that with the right parameter setup the GVNS soon manages to guide the search from arbitrary solutions into nearly optimal regions, Chapter 7 showed that usually at least some neighbors of the incumbent solution do have the same makespan and can thus be ignored during neighborhood evaluation. To this end, in this section we studied the impact of applying neighborhood pruning based on the critical set of jobs of the given incumbent solution in combination with purely *first-fit* step functions in the VND. Again, as a baseline algorithm the GVNS with piped VND, presented in Section 8.2, has been chosen and was modified for neighborhood pruning accordingly. As the approach primarily affects the efficiency of neighborhood evaluations, in the section we only consider instances with $n \geq 500$.

Table 8.15 show basic performance metrics for the described GVNS. Although for some skewed instances, a statistically significant improvement can be observed for $n = 2000$, compared to the approach based on range limitation in neighborhoods, presented previously, the improvement turned out to be only marginal nature. Even more, some tests have shown that the preparation of the critical set tends to be computationally too expensive compared to the devised neighbor evaluation scheme.

| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | Iterations | Exec. time [s] | p-value |
|---|---|---|---|---|---|---|---|---|
| B | 500 | 2 | 0.000 | <0.01 | 100.0 | 2.4 (± 2.89) | 0.5 (± 0.90) | - |
| B | 1000 | 2 | 0.000 | <0.01 | 100.0 | 3.5 (± 4.06) | 6.9 (± 6.09) | - |
| B | 2000 | 2 | 0.000 | <0.01 | 100.0 | 6.0 (± 6.77) | 58.2 (± 39.68) | - |
| B | 500 | 3 | 0.511 | 0.64 | 27.4 | 955.3 (± 652.19) | 682.1 (± 367.43) | $^{-}$**0.000** |
| B | 1000 | 3 | 0.467 | 0.47 | 10.2 | 234.3 (± 112.44) | 828.0 (± 222.77) | $^{-}$**0.000** |
| B | 2000 | 3 | 0.485 | 0.38 | 3.2 | 48.7 (± 24.47) | 880.8 (± 114.74) | $^{-}$**0.000** |
| B | 500 | 5 | <0.001 | <0.01 | 99.2 | 117.5 (± 317.81) | 38.4 (± 111.31) | 0.997 |
| B | 1000 | 5 | <0.001 | <0.01 | 94.4 | 114.6 (± 191.05) | 145.3 (± 241.14) | $^{+}$**0.002** |
| B | 2000 | 5 | <0.001 | <0.01 | 74.6 | 67.5 (± 81.31) | 335.4 (± 366.39) | $^{+}$**0.000** |
| S | 500 | 2 | 0.157 | 0.11 | 0.0 | 1375.7 (± 486.57) | 900.0 (± <0.01) | $^{-}$**0.000** |
| S | 1000 | 2 | 0.182 | 0.08 | 0.0 | 311.9 (± 98.86) | 900.0 (± <0.01) | $^{-}$**0.000** |
| S | 2000 | 2 | 0.274 | 0.10 | 0.0 | 74.3 (± 29.02) | 900.0 (± <0.01) | $^{+}$**0.000** |
| S | 500 | 3 | 0.016 | 0.04 | 20.6 | 759.8 (± 419.48) | 764.8 (± 288.12) | $^{-}$**0.000** |
| S | 1000 | 3 | 0.029 | 0.03 | 7.6 | 192.1 (± 82.39) | 856.3 (± 176.98) | $^{-}$**0.000** |
| S | 2000 | 3 | 0.089 | 0.09 | 1.6 | 39.1 (± 20.12) | 893.6 (± 56.46) | $^{+}$**0.000** |
| S | 500 | 5 | 0.024 | 0.04 | 27.0 | 370.8 (± 226.81) | 697.4 (± 351.76) | $^{-}$**0.001** |
| S | 1000 | 5 | 0.042 | 0.05 | 12.4 | 101.4 (± 53.06) | 819.4 (± 229.91) | 0.300 |
| S | 2000 | 5 | 0.281 | 0.22 | 1.2 | 8.5 (± 13.96) | 896.0 (± 42.66) | $^{+}$**0.000** |

Table 8.15: Performance metrics of a GVNS with randomized shaking, a piped VND and with a critical set based neighborhood pruning mechanism for balanced and skewed instances.

## 8.7 Comparison

This section finally compares the devised GVNS based algorithms with the approaches presented by Horn et al. [HRB19]. We refer to the algorithm from Section 8.2 as piped GVNS and to the range limited GVNS from Section 8.5 as RL-GVNS. Table 8.16 compares optimality gap, execution time and number of obtained optima of three GVNS variants against the A* algorithm as well as the CP/ILOG model of Horn et al. Among competing approaches where statistical significance was not apparent, again, *Wilcoxon ranksum* tests have been conducted with a significance level of 1%. Again, it is important to note that global optimality of a solution was verified with the lower bounds obtained by Horn et al. [HRB19].

Although the previous sections showed that there does exist some variance in the performance of different GVNS variants, the best performing approach on average manages to stay below 0.288% optimality gap for each instance class. However, as was also shown throughout this work and also by Horn et al. [HRB19], the performance varies significantly both within and between the considered instance classes.

Balanced instances, particularly with $m = 2$ can be solved to proven optimality for instances up to $n = 2000$ with each of the considered algorithms. Although our approaches starting from randomized initial solutions tend to be significantly slower than the A* based approach, the GVNS clearly benefits from the LLBH construction heuristic, allowing it to obtain globally optimal solutions on average already within 16.4 s for instances up to $n = 2000$.

Although for $m = 5$ a quite similar behavior can be observed, this does certainly not hold true for $m = 3$. While the GVNS based approaches still outperform the heuristic performance of CP/ILOG, a significant difference to A* with respect to both quality and temporal behavior can be observed. Even already for small or medium sized instances, as can be seen in Figure 8.10, the GVNS approaches show a relatively high variance, with some outliers even at %-gap $\approx 2\%$. Even various endeavors, like intensified shaking, failed to come significantly closer to solution qualities A* obtains almost instantly.

For skewed instances, on the other hand, generally a higher average execution time as well as a significantly lower number of encountered globally optimal solutions with an increasing instance size can be observed. Furthermore, Figure 8.10 shows that the variance of the optimality gap increases, suggesting a quite diverse difficulty in the considered instances. Despite this, GVNS based approaches significantly outperform the baselines throughout the considered instances. Particularly with an increasing number of secondary resources, the GVNS approaches tend to be relatively stable in the obtained solution quality, whereas the average solution quality obtained by A* is up to 0.915%.

More interestingly, it can be seen that the GVNS approaches also manage to obtain a higher number of globally optimal solutions for skewed instances. Although the number of optima for $m = 2$ is relatively low at only $\approx 40\%$ even for $n = 50$, as the number of secondary resources increases, still a significant number of optima can be obtained.

| | | | Piped-GVNS | | | | GVNS + LLBH | | | | RL-GVNS | | | | A* +BS+LS | | | | CP/ILOG | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| type | n | m | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | t[s] | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | t[s] | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | t[s] | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | t[s] | %-gap | $\sigma_{\%\text{-gap}}$ | %-opt | t[s] |
| B | 50 | 2 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | 1.1 | **0.000** | <0.01 | **100.0** | 0.0 |
| B | 100 | 2 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | 2.0 | **0.000** | <0.01 | **100.0** | 0.0 |
| B | 200 | 2 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | 5.4 | **0.000** | <0.01 | **100.0** | 0.1 |
| B | 500 | 2 | **0.000** | <0.01 | **100.0** | 2.8 | **0.000** | <0.01 | **100.0** | <0.1 | **0.000** | <0.01 | **100.0** | 2.0 | **0.000** | <0.01 | **100.0** | 35.3 | **0.000** | <0.01 | **100.0** | 1.3 |
| B | 1000 | 2 | **0.000** | <0.01 | **100.0** | 26.0 | **0.000** | <0.01 | **100.0** | 2.1 | **0.000** | <0.01 | **100.0** | 13.1 | **0.000** | <0.01 | **100.0** | 8.9 | **0.000** | <0.01 | **100.0** | 9.2 |
| B | 2000 | 2 | **0.000** | <0.01 | **100.0** | 223.1 | **0.000** | <0.01 | **100.0** | 16.4 | **0.000** | <0.01 | **100.0** | 83.7 | **0.000** | <0.01 | **100.0** | 46.3 | <0.001 | <0.01 | 98.0 | 63.5 |
| B | 50 | 3 | **0.044** | 0.20 | 91.2 | 89.5 | **0.043** | 0.19 | 91.4 | 83.0 | **0.050** | 0.22 | 91.2 | 86.1 | **0.017** | 0.08 | **96.0** | 1.1 | **0.068** | 0.30 | 92.0 | 0.0 |
| B | 100 | 3 | **0.106** | 0.28 | 81.6 | 182.7 | **0.100** | 0.26 | 80.6 | 183.7 | **0.112** | 0.29 | 79.6 | 192.8 | **0.021** | 0.09 | **92.0** | 2.0 | **0.226** | 0.55 | 78.0 | 4.2 |
| B | 200 | 3 | 0.180 | 0.46 | 75.4 | 250.0 | **0.150** | 0.40 | 79.2 | 200.4 | 0.176 | 0.45 | 74.0 | 257.8 | **0.016** | 0.06 | **92.0** | 5.9 | 0.556 | 1.12 | 56.0 | 319.4 |
| B | 500 | 3 | 0.251 | 0.41 | 50.2 | 475.6 | 0.117 | 0.24 | 64.2 | 328.8 | 0.260 | 0.42 | 47.0 | 499.8 | **<0.001** | <0.01 | **98.0** | 35.9 | 2.212 | 1.83 | 20.0 | 900.0 |
| B | 1000 | 3 | 0.203 | 0.32 | 32.6 | 647.8 | 0.059 | 0.17 | 74.6 | 233.9 | 0.216 | 0.33 | 31.0 | 644.6 | **0.001** | <0.01 | **98.0** | 6.1 | 3.094 | 1.46 | 2.0 | 899.9 |
| B | 2000 | 3 | 0.375 | 0.39 | 13.4 | 858.9 | 0.031 | 0.10 | 76.4 | 228.2 | 0.288 | 0.34 | 15.0 | 815.7 | **0.005** | 0.04 | **98.0** | 23.8 | 4.220 | 1.20 | 0.0 | 900.0 |
| B | 50 | 5 | **0.000** | <0.01 | **100.0** | 55.9 | **0.000** | <0.01 | **100.0** | 55.3 | **<0.001** | <0.01 | 99.4 | 60.3 | **0.000** | <0.01 | **100.0** | 1.2 | **0.000** | <0.01 | **100.0** | 0.7 |
| B | 100 | 5 | **0.000** | <0.01 | **100.0** | 56.4 | **0.000** | <0.01 | **100.0** | 55.8 | **0.000** | <0.01 | **100.0** | 55.6 | **0.000** | <0.01 | **100.0** | 2.2 | **0.000** | <0.01 | **100.0** | 9.5 |
| B | 200 | 5 | **0.000** | <0.01 | **100.0** | 40.6 | **0.000** | <0.01 | **100.0** | 36.2 | **0.000** | <0.01 | **100.0** | 40.1 | <0.001 | <0.01 | 98.0 | 6.5 | **0.000** | <0.01 | **100.0** | 91.3 |
| B | 500 | 5 | **<0.001** | <0.01 | 99.2 | 64.1 | **0.000** | <0.01 | **100.0** | 1.2 | **0.000** | <0.01 | **100.0** | 35.6 | **0.000** | <0.01 | **100.0** | 42.3 | <0.001 | <0.01 | 86.0 | 499.7 |
| B | 1000 | 5 | <0.001 | <0.01 | 89.0 | 297.0 | **0.000** | <0.01 | **100.0** | 5.5 | <0.001 | <0.01 | 96.0 | 130.9 | **0.000** | <0.01 | **100.0** | 7.9 | 0.359 | 0.12 | 0.0 | 900.0 |
| B | 2000 | 5 | <0.001 | <0.01 | 40.2 | 786.1 | **<0.001** | <0.01 | 99.4 | 20.6 | <0.001 | <0.01 | 86.6 | 302.7 | **0.000** | <0.01 | **100.0** | 30.4 | 0.478 | 0.14 | 0.0 | 900.0 |
| S | 50 | 2 | **0.157** | 0.23 | **42.0** | 524.1 | **0.157** | 0.23 | **42.0** | 523.5 | **0.163** | 0.23 | **42.0** | 527.3 | **0.268** | 0.38 | 40.0 | 11.4 | **0.210** | 0.28 | **42.0** | 899.9 |
| S | 100 | 2 | **0.165** | 0.31 | 34.4 | 605.8 | **0.166** | 0.31 | **34.8** | 605.5 | **0.172** | 0.32 | 33.8 | 613.0 | 0.367 | 0.49 | 26.0 | 44.8 | 0.323 | 0.47 | 12.0 | 900.0 |
| S | 200 | 2 | **0.101** | 0.17 | 14.2 | 808.8 | **0.102** | 0.17 | **15.2** | 799.9 | **0.111** | 0.18 | 14.8 | 807.8 | 0.440 | 0.33 | 2.0 | 65.2 | 0.642 | 0.51 | 0.0 | 900.0 |
| S | 500 | 2 | **0.102** | 0.08 | **0.2** | 900.0 | **0.096** | 0.08 | 0.0 | 900.0 | **0.095** | 0.08 | 0.0 | 900.0 | 0.532 | 0.18 | 0.0 | 88.7 | 2.736 | 0.51 | 0.0 | 900.0 |
| S | 1000 | 2 | 0.128 | 0.06 | **0.0** | 900.0 | **0.112** | 0.05 | **0.0** | 900.0 | **0.105** | 0.06 | **0.0** | 900.0 | 0.725 | 0.20 | **0.0** | 176.8 | 4.636 | 0.43 | **0.0** | 900.0 |
| S | 2000 | 2 | 0.315 | 0.12 | **0.0** | 900.0 | **0.154** | 0.05 | **0.0** | 900.0 | 0.214 | 0.11 | **0.0** | 900.0 | 0.786 | 0.18 | **0.0** | 252.7 | 4.784 | 0.39 | **0.0** | 900.0 |
| S | 50 | 3 | **0.034** | 0.15 | **82.2** | 197.9 | **0.035** | 0.15 | **82.2** | 196.8 | **0.035** | 0.15 | 82.0 | 198.6 | **0.053** | 0.21 | 82.0 | 1.3 | **0.035** | 0.15 | 80.0 | 27.7 |
| S | 100 | 3 | **0.030** | 0.10 | 82.0 | 180.8 | **0.030** | 0.10 | **83.6** | 172.4 | **0.030** | 0.10 | 82.8 | 177.6 | 0.153 | 0.37 | 50.0 | 16.5 | 0.060 | 0.15 | 52.0 | 899.7 |
| S | 200 | 3 | **0.025** | 0.11 | 76.8 | 272.4 | **0.024** | 0.11 | 76.4 | 270.1 | **0.025** | 0.11 | **78.8** | 240.0 | 0.117 | 0.26 | 34.0 | 26.4 | 0.135 | 0.21 | 36.0 | 899.8 |
| S | 500 | 3 | 0.008 | 0.02 | 25.2 | 738.4 | 0.008 | 0.02 | 26.4 | 728.5 | **0.006** | 0.02 | **42.4** | 598.3 | 0.177 | 0.24 | 14.0 | 121.6 | 1.360 | 0.76 | 4.0 | 900.0 |
| S | 1000 | 3 | 0.018 | 0.02 | 6.8 | 857.9 | 0.020 | 0.02 | 5.8 | 870.4 | **0.009** | 0.02 | **19.2** | 776.0 | 0.621 | 0.47 | 2.0 | 248.0 | 2.872 | 0.93 | 0.0 | 900.0 |
| S | 2000 | 3 | 0.114 | 0.11 | 0.4 | 898.3 | 0.244 | 0.18 | 0.0 | 900.0 | **0.041** | 0.05 | **5.8** | 877.4 | 0.701 | 0.41 | 0.0 | 480.2 | 4.296 | 0.98 | 0.0 | 900.0 |
| S | 50 | 5 | **0.046** | 0.14 | 83.7 | 147.2 | **0.046** | 0.14 | 83.7 | 147.1 | **0.046** | 0.14 | 83.7 | 147.1 | **0.077** | 0.19 | 80.0 | 1.4 | **0.045** | 0.14 | **84.0** | 15.0 |
| S | 100 | 5 | **0.006** | 0.02 | 88.4 | 142.0 | **0.006** | 0.02 | **88.6** | 138.6 | **0.006** | 0.02 | 88.4 | 139.2 | 0.064 | 0.18 | 66.0 | 6.1 | 0.019 | 0.04 | 70.0 | 899.6 |
| S | 200 | 5 | **0.034** | 0.14 | 73.0 | 314.8 | **0.034** | 0.14 | 72.2 | 309.9 | **0.034** | 0.14 | **77.2** | 252.1 | 0.281 | 0.49 | 34.0 | 38.8 | 0.161 | 0.25 | 28.0 | 900.0 |
| S | 500 | 5 | 0.014 | 0.03 | 30.0 | 688.3 | 0.015 | 0.03 | 26.4 | 722.3 | **0.009** | 0.02 | **46.8** | 555.3 | 0.347 | 0.34 | 16.0 | 188.6 | 1.229 | 0.95 | 8.0 | 899.9 |
| S | 1000 | 5 | 0.035 | 0.04 | 6.4 | 868.6 | 0.041 | 0.04 | 2.6 | 891.0 | **0.012** | 0.02 | **22.2** | 765.6 | 0.702 | 0.50 | 0.0 | 387.3 | 2.478 | 1.11 | 0.0 | 900.0 |
| S | 2000 | 5 | 0.393 | 0.26 | 0.0 | 900.1 | 0.777 | 0.36 | 0.0 | 900.1 | **0.105** | 0.10 | **2.0** | 892.8 | 0.915 | 0.54 | 0.0 | 789.3 | 4.229 | 1.22 | 0.0 | 900.0 |

Table 8.16: Comparison of GVNS based algorithms, A* and CP/ILOG for balanced and skewed instances.

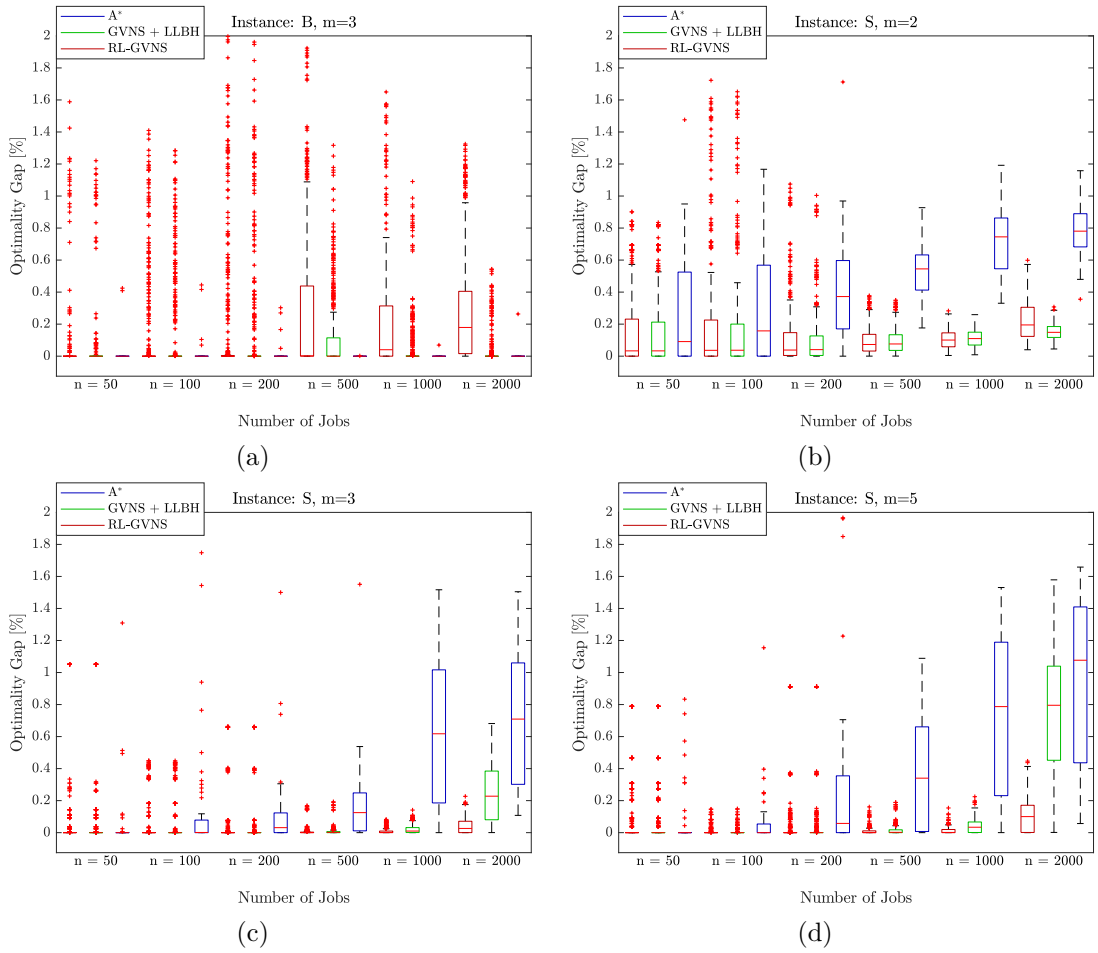



Figure 8.10: Comparison of the distribution of the optimality gap of solutions obtained by GVNS+LLBH, RL-GVNS and A* for balanced and skewed instances.

## 8.8 Summary

This chapter showed a study of different GVNS variants for the JSOCMSR. Based on a set of experiments, various behavioral aspects of the considered algorithms have been analyzed and compared to each other. We started with the study of a GVNS employing a sequential VND scheme that turned out as problematic as the instance size increases. By analyzing success rates of neighborhood structures during the search, it turned out that effectiveness of VND neighborhoods varies over time, suggesting alternative VND progression schemes to be studied.

Based on this, a piped VND scheme showed its effectiveness even for large instances and subsequently served as baseline for further GVNS variants. Subsequently, the impact of construction heuristics and neighborhood reduction approaches was investigated.

Construction heuristics, on the one hand, turned out as effective for balanced instances, where a significant improvement could be obtained particularly in hard instances with three secondary instances, while for some skewed instances, they seemed to force the search into a rather suboptimal region right from the very beginning.

Neighborhood reduction techniques, on the other hand, showed their effectiveness particularly in early phases of the search, where a high gradient indicated large improvements in rather short periods of the search.

Finally, in Section 8.7 the most promising GVNS approaches were compared against baselines from Horn et al. [HRB19]. Although it was shown that for some particularly hard balanced instances our GVNS approaches did not come even close to objectives obtained by the A* algorithm, the converse was true for skewed instances. For any of the considered skewed instances classes, the GVNS achieved on average the lowest optimality gap, while at the same time obtained the most globally optimal solutions.

CHAPTER 9

# Conclusion

This thesis dealt with a VNS for the JSOCMSR problem, a combinatorial optimization problem appearing in the context of patient scheduling in cancer treatment facilities. Although recent exact solution techniques devised by Horn et al. [HRB19] showed their efficiency for both real-world and artificially generated instances, there was still some room for heuristic improvement for some types of instances. To this end, an alternative solution method based on VNS was devised and thoroughly analyzed.

In the first part of this work, we focused on fundamental mechanisms for an efficient evaluation of neighborhoods explored during the search. Based on exploiting synchronized offsets of jobs in the incumbent solution and the respective neighbor solution, two evaluation schemes were considered and studied in detail.

Extensive experimental evaluation showed the practical relevance of both approaches. Experiments showed that for the considered instances the computation time for one of these evaluation schemes is even independent w.r.t. the number of jobs. Although it was also shown that the synchronization technique can be applied to instances of reasonable size more than 90% of the time, the approach may suffer from severe performance degradation in highly imbalanced instances, where some solutions may inevitably lead to a worst-case execution time of $\mathcal{O}(nm)$. To this end, a more comprehensive study on the efficiency of the implemented evaluation scheme, particularly on a more diverse set of instance types should be conducted in future work and alternative evaluation schemes could be considered.

Based on the described evaluation schemes, different standard neighborhood structures were then selected, implemented and experimentally analyzed. It was shown that search landscapes for the studied neighborhood structures tend to be relatively rugged, i.e. inducing a relatively low correlation of the objective values of neighboring solutions, which manifests itself in significant changes in the objective, even for minor modifications in the solution representation.

Another important part of the landscape analysis consisted of a study of fundamental characteristics of local optima to study their suitability for the JSOCMSR. Experiments showed a relatively clear hierarchy of the neighborhood structures in terms of solution quality, in addition to complementing characteristics that favor their applicability in VNS.

In the course of the landscape analysis we then concentrated also on globally optimal solutions. By analyzing characteristics of a large set of global optima for relatively small instances, we aimed to analyze at least some factors affecting the *difficulty* of an instance. It was shown that even for small instances a vast amount of distinct global optima can be obtained, likely also due to symmetries in an instances structure. Even more, by comparing average distances in solution populations, it was further shown that global optima tend to be widely distributed in the search space, although they still share some structural characteristics with respect to the jobs constituting their critical set. It is, however, of highest importance that this analysis was only a first step towards a thorough analysis of the considered instances, and much is left for future work to study various characteristics in more depth.

The last part of this work was dedicated to an experimental comparison of different GVNS based approaches to the anytime A* baseline algorithm from Horn et al. [HRB19]. In the evaluation we primarily focused on comparing the optimality gap for two different types of instances. We showed that our approaches generally are capable to obtain solution qualities comparable to A*, where throughout the considered instance classes an average optimality gap below 0.288% was achieved.

Balanced instances, on the one hand, showed a quite diverse behavior. While some instances turned out as almost trivial, instances with $m = 3$ secondary resources showed to be more challenging for the GVNS. Some instances that could be solved to proven optimality by A* instantly, were particularly hard for the GVNS and only starting the GVNS from initial solutions obtained by the LLBH construction heuristic enabled our approach to come at least closer to the results of Horn et al.

For skewed instances, on the other hand, GVNS based approaches were able to significantly improve the average solution quality compared to solutions obtained from A*. Moreover, it was possible to show for more instances the proven optimality by means of lower bounds obtained from A*. However, going in accordance with the findings of Horn et al., in some of these instances global optima were particularly rare or hard to obtain, suggesting further room for heuristic improvement. To this end, future work may concentrate on larger or more complex neighborhood structures that may be combined with the standard neighborhood structures selected in this work.

Finally, as the variance in solution quality among the considered instance set showed to be relatively high, future work may also concentrate on a more concise, but elite set of instances.

# List of Figures


1.1 Phases of a JSOCMSR job . . . . . 2
1.2 JSOCMSR example solution . . . . . 3

4.1 Examples of search space landscapes . . . . . 25

5.1 Solution of a JSOCMSR instance described by a permutation . . . . . 30
5.2 Illustration of a synchronization border . . . . . 32
5.3 Range-based re-evaluation example. . . . . 33
5.4 Example of an insertion move. . . . . 35
5.5 Example of an exchange move. . . . . 36
5.6 Example of an inversion move. . . . . 37
5.7 Job interleaving example . . . . . 38
5.8 Comparison of utilization gaps on the common resource in a random and a high-quality solution. . . . . 39
5.9 Example of critical jobs in a solution. . . . . 40
5.10 Dependency graph of a solution . . . . . 41
5.11 Intensified shaking with randomized moves. . . . . 46
5.12 Subproblem based shaking . . . . . 48

6.1 Comparison of the synchronization distance in balanced and skewed instances. 54
6.2 Comparison of the synchronization hit-rate in different instances. . . . . 54
6.3 Comparison of the execution times of evaluation schemes for different neighborhood structures . . . . . 56

7.1 Comparison of the correlation length of different neighborhood structures 60
7.2 Comparison of the correlation length of different neighborhood structures with different move-range limitations . . . . . 61
7.3 Comparison of the average optimality gap of local optima obtained by different neighborhood structures. . . . . 63
7.4 Comparison of the average depth of local optima obtained by different neighborhood structures. . . . . 64
7.5 Comparison of the relative improvement of a single step in local search procedures. . . . . 65
7.6 Global optimum of a balanced instance with two secondary resources. . . 69

7.7 Global optimum of a skewed instance with two secondary resources. . . . 69

8.1 Optimality gap over time for the GVNS with randomized shaking. . . . . 79
8.2 Distribution of the average number of successful moves of different VND neighborhood structures. . . . . 80
8.3 Total number of improving move operations in the VND over time for GVNS with randomized shaking. . . . . 81
8.4 Comparison of the optimality gap over time for the GVNS with randomized shaking and the GVNS with piped VND scheme. . . . . 83
8.5 Optimality gap over time for the GVNS with randomized sharking and a piped VND scheme. . . . . 84
8.6 Comparison of success rates of shaking neighborhood structures of purely randomized shaking and intensified shaking. . . . . 87
8.7 Optimality gap over time for the GVNS with a piped VND starting from solutions obtained by LLBH. . . . . 90
8.8 Optimality gap over time for the GVNS making use of range limited neighborhoods. . . . . 93
8.9 Distribution of successful moves of different VND neighborhood structures of the GVNS employing range limited neighborhood structures in the VND. 95
8.10 Comparison of the distribution of the optimality gap of solutions obtained by GVNS+LLBH, RL-GVNS and A* for balanced and skewed instances. . . 99

# List of Tables


5.1 Example of interleaved shaking neighborhoods . . . . . 44

7.1 Complementing characteristics of different neighborhoods structures. . . . 66
7.2 Comparison of characteristics of global optima . . . . . 67
7.3 Metrics describing the workload distribution in different instances . . . . 68
7.4 Comparison of the average normalized distances of a set of global optima and a set of random solutions . . . . . 70
7.5 Critical set metrics of global optima . . . . . 71
7.6 Critical set metrics of local optima . . . . . 72

8.1 Neighborhood structures for the VND of the GVNS with purely randomized shaking. . . . . 77
8.2 Shaking neighborhood structures of the GVNS with purely randomized shaking. . . . . 78
8.3 Performance metrics of a GVNS with randomized shaking for balanced instances. . . . . 78
8.4 Performance metrics of a GVNS with randomized shaking for skewed instances. 80
8.5 Performance metrics of a GVNS with randomized shaking and a piped VND for balanced instances. . . . . 82
8.6 Performance metrics of a GVNS with randomized shaking and a piped VND for skewed instances. . . . . 85
8.7 Shaking neighborhood structures of the GVNS with intensified shaking. . 86
8.8 Performance metrics of a GVNS with intensified randomized shaking and a piped VND for balanced instances. . . . . 86
8.9 Performance metrics of a GVNS with intensified randomized shaking and a piped VND for skewed instances. . . . . 88
8.10 Performance metrics of a GVNS with randomized shaking, a piped VND and starting with initial solutions obtained with LLBH for balanced instances. 89
8.11 Performance metrics of a GVNS with randomized shaking, a piped VND and starting with initial solutions obtained with LLBH for skewed instances. . . 91
8.12 Neighborhood structures for the VND of the GVNS with range-limited neighborhood structures. . . . . 92
8.13 GVNS and with a range-limited VND for balanced instances . . . . . 92

8.14 GVNS and with a range-limited VND for skewed instances . . . . . 94
8.15 Performance metrics of a GVNS with randomized shaking, a piped VND and with a critical set based neighborhood pruning mechanism for balanced and skewed instances. . . . . 96
8.16 Comparison of GVNS based algorithms, A* and CP/ILOG for balanced and skewed instances. . . . . 98

# List of Algorithms


3.1 Local-Search . . . . . 14
3.2 Variable Neighborhood Descent . . . . . 20
3.3 General Variable Neighborhood Search . . . . . 21

5.1 JSOCMSR solution decoding procedure . . . . . 34
5.2 Procedure to prepare a vector of probabilities for intensified shaking. . 46
5.3 Intensified shaking by a destroy and repair like mechanism. . . . . 47
5.4 General Variable neighborhood Search for the JSOCMSR . . . . . 49
5.5 VND for the JSOCMSR . . . . . 49

# Acronyms

**BVNS** basic variable neighborhood search. 22

**FNS** fixed neighborhood search. 21

**GVNS** general variable neighborhood search. 22, 45

**ILS** iterated local search. 18, 21

**JSOCMSR** Job Sequencing with One Common and Multiple Secondary Resources. 1–4, 7, 10, 11, 25–28, 31–33, 35, 36, 38, 40, 42–45, 55, 60, 61, 64, 65, 68, 70, 71, 81

**LINAC** Linear Accelerator. 1, 9

**LNS** large neighborhood search. 16

**RVNS** reduced variable neighborhood search. 22

**TS** tabu search. 18

**TSP** traveling salesman problem. 10, 15

**VND** variable neighborhood descent. 20–22, 37, 39, 45, 60, 62, 64, 69, 85

**VNDS** variable neighborhood decomposition search. 23

**VNS** Variable Neighborhood Search. 3–5, 16, 18, 20–23, 26, 31, 38, 43–45, 60, 63, 65

# Bibliography


[AFNP11] Alessandro Agnetis, Marta Flamini, Gaia Nicosia, and Andrea Pacifici. A job-shop problem with one additional resource type. *Journal of Scheduling*, 14(3):225–237, 2011.

[AL97] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, USA, 1st edition, 1997.

[All15] Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, 2015.

[All16] Ali Allahverdi. A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665–686, 2016.

[AOS00] Ravindra K. Ahuja, James B. Orlin, and Dushyant Sharma. Very large-scale neighborhood search. *International Transactions in Operational Research*, 7(4):301–317, 2000.

[AZ00] Eric Angel and Vassilis Zissimopoulos. On the classification of np-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99(1):261–277, 2000.

[BHMT00] Jack Brimberg, Pierre Hansen, Nenad Mladenović, and Eric D. Taillard. Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem. *Operations Research*, 48(3):444–460, 2000.

[BKM94] Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101–113, 1994.

[BMU15] Jack Brimberg, Nenad Mladenović, and Dragan Urošević. Solving the maximally diverse grouping problem by skewed general variable neighborhood search. *Information Sciences*, 295:650–675, 2015.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Compututing Surveys*, 35(3):268–308, 2003.

[Cay49] A. Cayley. LXXVII. Note on the theory of permutations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232):527–529, 1849.

[CDB10] Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, 201(3):921–932, 2010.

[Čer85] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[CGG64] P C. Gilmore and Ralph Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, 1964.

[CLLR03] B Cheang, H Li, A Lim, and B Rodrigues. Nurse rostering problems—a bibliographic survey. *European Journal of Operational Research*, 151(3):447–460, 2003.

[Con00] Richard K. Congram. *Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation*. PhD thesis, University of Southampton, 2000.

[CP12] Elkin Castro and Sanja Petrovic. Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. *Journal of Scheduling*, 15(3):333–346, 2012.

[Cro58] AG Croes. A method for solving traveling-salesman problems. *Operations Research*, 6:791–812, 1958.

[CV09] Tugba Cayirli and Emre Veral. Outpatient scheduling in health care: A review of literature. *Production and Operations Management*, 12:519–549, 2009.

[DB05] Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2):243–278, 2005.

[dBS01] Matthijs den Besten and Thomas Stützle. Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. In *Proceedings of the 4th Metaheuristics International Conference (MIC)*, pages 545–549, Porto, Portugal, 2001.

[DD99] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, volume 2, pages 1470–1477 Vol. 2, Washington, DC, USA, 1999.

[DGS06] Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1):65–89, 2006.

[DH98] Michael Deza and Tayuan Huang. Metrics on permutations, a survey. *Journal Of Combinatorics, Information and System Sciences*, 23(1–4):173–185, 1998.

[Dor92] Marco Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, 1992.

[DS10] Marco Dorigo and Thomas Stützle. Ant Colony Optimization: Overview and Recent Advances. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 227–263. Springer, 2nd edition, 2010.

[EJKS04] Andreas Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153:3–27, 2004.

[ES03] A Eiben and Jim Smith. *Introduction To Evolutionary Computing*, volume 45 of *Natural Computing Series*. Springer, 2003.

[FCS97] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[FOW66] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, 1966.

[FR89] Thomas A Feo and Mauricio G.C Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[GH11] Fred Glover and Jin-Kao Hao. The case for strategic oscillation. *Annals of Operations Research*, 183(1):163–173, 2011.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[GP10] Michel Gendreau and Jean-Yves Potvin. Tabu Search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 41–59. Springer, 2nd edition, 2010.

[GSS00] Celia Glass, Yakov Shafransky, and Vitaly Strusevich. Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics*, 47:304–328, 2000.

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[Han86] Pierre Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Proceedings of Congress on numerical methods in combinatorial optimization*, pages 70–145, Capri, Italy, 1986.

[HB10] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[HJdA94] Alain Hertz, Brigitte Jaumard, and Marcus Poggi de Aragão. Local optima topology for the k-coloring problem. *Discrete Applied Mathematics*, 49(1):257–280, 1994. Special Volume Viewpoints on Optimization.

[HJMP00] P Hansen, B Jaumard, N Mladenovic, and A Parreira. Variable neighborhood search for weighted maximum satisfiability problem. *Les Cahiers du GERAD*, 62, 2000.

[HKT95] T. C. Hu, Andrew B. Kahng, and Chung-Wen Albert Tsao. Old bachelor acceptance: A new class of non-monotone threshold accepting methods. *Journal on Computing, Operations Research Society of America (ORSA)*, 7(4):417–425, 1995.

[HM06] Pierre Hansen and Nenad Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006. IV ALIO/EURO Workshop on Applied Combinatorial Optimization.

[HMM10] Pierre Hansen, Nenad Mladenović, and José A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[HMP01] Pierre Hansen, Nenad Mladenovic, and Dionisio Pérez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

[HMTH17] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[Hol75] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[HPS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, volume 32. Prentice-Hall, Inc., 1982.

[HPS00] Nicholas G. Hall, Chris N. Potts, and Chelliah Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102(3):223–243, 2000.

[HRB17] Matthias Horn, Günther R. Raidl, and Christian Blum. Job sequencing with one common and multiple secondary resources: A problem motivated from particle therapy for cancer treatment. In *Machine Learning, Optimization, and Big Data - Third International Conference, MOD 2017*, volume 10710 of *LNCS*, pages 506–518, 2017.

[HRB19] Matthias Horn, Günther Raidl, and Christian Blum. Job sequencing with one common and multiple secondary resources: An A*/beam search based anytime algorithm. *Artificial Intelligence*, 277:103173, 2019.

[HRR19] Matthias Horn, Günther Raidl, and Elina Rönnberg. A* search for prize-collecting job sequencing with one common and multiple secondary resources. Technical Report AC-TR-19-002, Algorithms and Complexity Group, TU Wien, 2019.

[HS87] J.Pirie Hart and Andrew W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

[HS04] Holger Hoos and Thomas Sttzle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., 2004.

[Ing96] Lester Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control Cybernetics*, 25:33–54, 1996.

[JWH04] Junlin Chang, Weiwu Yan, and Huihe Shao. Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times. In *Proceedings of the 2004 American Control Conference*, volume 2, pages 1412–1416, Boston, MA, USA, 2004.

[Kau93] S.A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[KBDCVBVL04] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7:441–499, 2004.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, WA, Australia, 1995.

[Ken38] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[KP09] Truword Kapamara and Dobrila Petrovic. A heuristics and steepest hill climbing method to scheduling radiotherapy patients. In *Proceedings of the International Conferenceon Operational Research Applied to Health Services (ORAHS)*, Catholic University of Leuven, Leuven, Belgium, 2009.

[Lar93] S.N. Larsson. Radiotherapy patient scheduling using a desktop personal computer. *Clinical Oncology*, 5(2):98–101, 1993.

[LIDLP+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Lin65] S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965.

[LK73] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[LOT10] H. R. Lourenco, Martin O., and Stützle T. Iterated local search: Framework and Applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 363–397. Springer, 2nd edition, 2010.

[MC10] Pablo Moscato and Carlos Cotta. A Modern Introduction to Memetic Algorithms. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 141–183. Springer, 2nd edition, 2010.

[MD19] Joren Marynissen and Erik Demeulemeester. Literature review on multi-appointment scheduling problems in hospitals. *European Journal of Operational Research*, 272(2):407–419, 2019.

[MHRR17] Johannes Maschler, Thomas Hackl, Martin Riedler, and Günther R. Raidl. An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *Proceedings of the 12th Metaheuristics International Conference*, pages 465–474, Barcelona, Spain, 2017.

[Mla95] N. Mladenović. A variable neighborhood algorithm – a new metaheuristics for combinatorial optimization. In *Abstracts of Papers Presented at Optimization Days. Montreal*, 1995.

[MR18] Johannes Maschler and Günther R. Raidl. Particle therapy patient scheduling with limited starting time variations of daily treatments. *International Transactions in Operational Research*, 2018.

[MR19] Johannes Maschler and Günther Raidl. Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources. Technical Report AC-TR-19-003, Algorithms and Complexity Group, TU Wien, 2019.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[MRSR16] Johannes Maschler, Martin Riedler, Markus Stock, and Günther R Raidl. Particle therapy patient scheduling: First heuristic approaches. In *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling*, pages 223–244, Udine, Italy, 2016.

[NI98] Koji Nonobe and Toshihide Ibaraki. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, 106(2):599–623, 1998.

[NZ96] E. Nowicki and S. Zdrzałka. Single machine scheduling with major and minor setup times: A tabu search approach. *Journal of the Operational Research Society*, 47(8):1054–1064, 1996.

[Pin08] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition, 2008.

[PL08] S. Petrovic and P. Leite-Rocha. Constructive and grasp approaches to radiotherapy treatment scheduling. In *Advances in Electrical and*

*Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 192–200, 2008.

[PLSS06] Sanja Petrovic, William Leung, Xueyan Song, and Santhanam Sundar. Algorithms for radiotherapy treatment booking. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'2006)*, pages 105–112, 2006.

[PMP09] Dobrila Petrovic, Mohammad Morshed, and Sanja Petrovic. Genetic algorithm based scheduling of radiotherapy treatments for cancer patients. In Carlo Combi, Yuval Shahar, and Ameen Abu-Hanna, editors, *Artificial Intelligence in Medicine*, pages 101–105. Springer, 2009.

[Rec70] Ingo Rechenberg. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, TU Berlin, 1970.

[Ree99] C.R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86(0):473–490, 1999.

[RS65] Stanley Reiter and Gordon Sherman. Discrete optimizing. *Siam Journal on Applied Mathematics - SIAMAM*, 13(3):864–889, 1965.

[RS07] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[SÏ3] Kenneth Sörensen. Metaheuristics – the metaphor exposed. *International Transactions of Operations Research*, 22:3–18, 2013.

[SG13] Kenneth Sörensen and Fred W. Glover. Metaheuristics. In Saul I. Gass and Michael C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer, 2013.

[SH83] Dileep R. Sule and Kin Yann Huang. Sequency on two and three machines with setup, processing and removal times separated. *International Journal of Production Research*, 21(5):723–732, 1983.

[Sia16] P. Siarry. *Metaheuristics.* Springer, 2016.

[SS92] Peter F. Stadler and Wolfgang Schnabl. The landscape of the traveling salesman problem. *Physics Letters A*, 161(4):337–344, 1992.

[SS05a] Tommaso Schiavinotto and Thomas Stützle. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, 2005.

[SS05b] Marc Sevaux and Kenneth Sörensen. Permutation distance measures for memetic algorithms with population management. *Proceedings of 6th Metaheuristics International Conference (MIC'05)*, 94:832–838, 2005.

[SS07] Tommaso Schiavinotto and Thomas Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153, 2007.

[Sta96] Peter F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20(1):1–45, 1996.

[Sul82] Dileep R. Sule. Sequencing n jobs on two machines with setup, processing and removal times separated. *Naval Research Logistics Quarterly*, 29(3):517–519, 1982.

[Tal09] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.

[TMM+17] Raca Todosijevic, Anis Mjirda, Marko Mladenovic, Saïd Hanafi, and Bernard Gendron. A general variable neighborhood search variants for the travelling salesman problem with draft limits. *Optimization Letters*, 11(6):1047–1056, 2017.

[TO+89] Paul Michael Thompson, James B Orlin, et al. The theory of cyclic transfers. *Operations Research Center Working Papers*, 1989.

[VBD18] Petra Vogl, Roland Braune, and Karl F. Doerner. A multi-encoded genetic algorithm approach to scheduling recurring radiotherapy treatment activities with alternative resources, optional activities, and time window constraints. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2017*, pages 373–382. Springer, 2018.

[VBD19] Petra Vogl, Roland Braune, and Karl F. Doerner. Scheduling recurring radiotherapy appointments in an ion beam facility. *Journal of Scheduling*, 22(2):137–154, 2019.

[vdVWZ98] Jack A. A. van der Veen, Gerhard J. Woeginger, and Shuzhong Zhang. Sequencing jobs that require common resources on a single machine: A solvable case of the tsp. *Mathematical Programming*, 82(1):235–254, 1998.

[VFSB17] Jonas Volland, Andreas Fügener, Jan Schoenfelder, and Jens O. Brunner. Material logistics in hospitals: A literature review. *Omega*, 69:82–101, 2017.

[VHvVV+16] Bruno Vieira, Erwin W. Hans, Corine van Vliet-Vroegindeweij, Jeroen van de Kamer, and Wim van Harten. Operations research for resource planning and -use in radiotherapy: a literature review. *BMC Medical Informatics and Decision Making*, 16(1):149, 2016.

[Wat10] Jean-Paul Watson. An Introduction to Fitness Landscape Analysis and Cost Models for Local Search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 599–623. Springer, 2nd edition, 2010.

[Wei90] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336, 1990.

[Whi13] Darrell Whitley. *Sharpened and Focused No Free Lunch and Complexity Theory*, pages 451–476. Springer, 2013.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[Wri32] Sewall Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. *Proceedings of the Sixth International Congress of Genetics*, 1:356–366, 1932.

[WSB76] M.S Waterman, T.F Smith, and W.A Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.

[YH79] Teruhiko Yoshida and Katsundo Hitomi. Optimal two-stage production scheduling with setup times separated. *A I I E Transactions*, 11(3):261–263, 1979.