Pontifícia Universidade Católica
do Rio de Janeiro

Davi Zerpini Mecler

A Metaheuristic for the Pipe Laying Support Vessels Scheduling Problem

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Engenharia de Produção of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia de Produção.

Advisor : Prof. Rafael Martinelli Pinto
Co-advisor: Prof. Arild Hoff

Rio de Janeiro
April 2020

Pontifícia Universidade Católica
do Rio de Janeiro

Davi Zerpini Mecler

A Metaheuristic for the Pipe Laying Support Vessels Scheduling Problem

Dissertation presented to the Programa de Pós–graduação em Engenharia de Produção of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia de Produção. Approved by the Examination Committee.

Prof. Rafael Martinelli Pinto
Advisor
Departamento de Engenharia Industrial – PUC-Rio

Prof. Arild Hoff
Co-advisor
Molde University College – HiMolde

Prof. Silvio Hamacher
Departamento de Engenharia Industrial – PUC-Rio

Prof. Thibaut Victor Gastón Vidal
Departamento de Informática – PUC-Rio

Rio de Janeiro, April 24, 2020

Davi Zerpini Mecler

Bachelor in Industrial Engineering (2017) at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Earned a scholarship by CNPq for the masters. Worked at a state logistics company before entering the PUC-Rio masters program. The author works as a researcher at Tecgraf Institute since June 2019, supporting decision makers in the Oil & Gas Industry.

Bibliographic data

Mecler, Davi Zerpini

A Metaheuristic for the Pipe Laying Support Vessels Scheduling Problem / Davi Zerpini Mecler; advisor: Rafael Martinelli Pinto ; co-advisor: Arild Hoff. - 2020.
75 f.: il. color. ; 30 cm

Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Industrial, 2020.
Inclui bibliografia

1. Engenharia Industrial - Teses. 2. Escalonamento de Máquinas Paralelas Idênticas. 3. Family Setup Times. 4. Iterated Local Search. 5. Cadeia de Suprimentos de Óleo e Gás. I. Martinelli Pinto, Rafael. II. Hoff, Arild. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Industrial. IV. Título.

CDD: 658.5

Acknowledgments

Many people and institutions have contributed for this work development and I will use this space to mention and thank all of them. First I want to thank my advisor Rafael Martinelli, for all the knowledge, support and friendship he gave me through all this thesis development, spending many hours in discussions and suggestions for the improvement of this work. Silvio Hamacher who, together with Rafael Martinelli, proposed to me this thesis theme and invited me to work at Tecgraf Institute, where I learn very much everyday and flourished the knowledge described in this work. My coadvisor Arild Hoff and professor Irina Gribkovskaia, who received me in Norway, at Molde University College, where I learned a lot, and helped me to improve my skills and this thesis text.

I want to thank all my colleagues in Tecgraf, who inspire me every day, creating a motivating working environment, helping me to evolve as student and human being. Some of these colleagues have direct contribution on this study and I must cite them here. Gabriela Ribas and Pedro Lobato, who coordinate the project I am inserted, intimately related to this thesis development, Victor Abu and Janaina Marchesi, who have taught me many skills in optimization methods and programming, being crucial for this thesis evolution, Iuri Santos, Luana Carrilho and Rachel Ventriglia, who also help me to improve as a researcher.

I must thank my family, Ian Mecler, Elizabeth Zerpini, Jordana Mecler, Katia Mecler, Rosinha Goldenstein, Nair Semblano e Edir Semblano for all the love and support I have received through all my life. Jordana Mecler in special helped me through this thesis, as she is an engineer colleague and taught me many programming skills.

This work was supported by PUC-Rio, Tecgraf Institute and the studied company, providing an extremely healthy environment for the development of this thesis. I would like to thank some of my professors: Hugo Repolho, who first invited me to enter the Industrial Engineer masters program, Mario Bittencourt, who taught me very much about logistics and transportation, Fernando Cyrino, Luciana Pessoa, Julia Fleck, Felipe Scavarda and José Eugenio, for all the transmitted knowledge. Also I must mention my masters and Tecgraf colleagues: Thiago Augusto, who is a great friend and student, Isabella Fischer, João Gelli, Bruno Mariani, Luiza Fiorencio, Danuza Dreux, Igor Peres, Leonardo Bastos, Bianca Brandão, Leila Dantas, Amanda Batista, Adrian Perez, Guilherme Faveret, Ricardo Terzian and Samir Azzan.

At last I want to thank my friends Miguel, Nicolas, Edgard, Arthur, Rafael, Lucca, Daniel, Beatriz, Gabriel, Pedro, Rodrigo, Luis Filipe, João Felipe, Tom and many others.

This study was financed in part by the Coordenação de Aperfeiçoamento de

Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and I am very grateful for this support.
Also I would like to thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), who provided my master's program scholarship and PUC-Rio where I have lived great moments and learned more than I could imagine.

Abstract

Mecler, Davi Zerpini; Martinelli Pinto, Rafael (Advisor); Hoff, Arild (Co-Advisor). **A Metaheuristic for the Pipe Laying Support Vessels Scheduling Problem**. Rio de Janeiro, 2020. 75p. Dissertação de Mestrado – Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro.

This work objective is to propose an Iterated Local Search metaheuristic to minimize the weighted completion time of jobs on identical parallel machines scheduling problems. The secondary objective of this work is to propose a practical solution to the real problem of the studied company. The motivation of this work consists on a practical application in the Oil & Gas industry, where the PLSV vessels perform ordered operations in a set of wells aiming to maximize the oil production. The characteristics of the problem such as: eligibility between vessels and operations, setup times related to the family of activities, association between operations and wells and setup times depending on the material arrival fit well the identical parallel machine schedule modelling. In this work, an introduction about the theme is presented, followed by a literature review on identical parallel machine scheduling problems, the mathematical formulation with the problem description is presented, the methodology, including the solution representation, constructive heuristic, neighborhood structures, local search and Iterated Local Search is exposed, at last, the method results and conclusions of the work are summarized.

Keywords

Metaheuristic Parallel Machine Scheduling Oil & Gas PLSV Iterated Local Search

Resumo

Mecler, Davi Zerpini; Martinelli Pinto, Rafael; Hoff, Arild. **Uma Metaheuristica para o Problema de Escalonamento de Pipe Laying Support Vessels**. Rio de Janeiro, 2020. 75p. Dissertação de Mestrado – Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho tem como objetivo propor uma metaheurística Iterated Local Search para minimizar o tempo de término ponderado de jobs em problemas de escalonamento de máquinas idênticas paralelas. O objetivo secundário do trabalho é propor uma solução prática para um problema real da companhia estudada em questão. A motivação para o trabalho consiste em uma aplicação prática na indústria de óleo e gás, onde os navios PLSV realizam operações em poços de forma ordenada e visando a antecipação dos poços de petróleo mais produtivos. As características do problema, tais quais: elegibilidade entre navio e operações, tempos de setup relativos à família de atividades, associações entre operações e poços e setups que dependem da chegada de material se adequam a modelagem baseada em escalonamento de máquinas paralelas idênticas. No trabalho uma introdução à respeito do tema é apresentada, em seguida é feita uma revisão da literatura sobre problemas de máquinas paralelas idênticas, a formulação matemática com a descrição do problema é apresentada, a metodologia contemplando representação da solução, heuristica construtiva, vizinhanças exploradas, busca local e Iterated Local Search é exposta, por fim são apresentados os resultados do método e as conclusões sobre o trabalho.

Palavras-chave

Metaheurística Escalonamento de Máquinas Paralelas Óleo & Gás PLSV Iterated Local Search

Table of contents

1 Introduction **11**

2 Literature Review **16**

3 Mathematical Formulation **21**
3.1 Problem Description 21
3.2 Positional Scheduling Formulation 23

4 Methodology **26**
4.1 Solution Representation And Evaluation 26
4.2 Constructive Heuristic 27
4.2.1 Constructive Heuristic Decision 32
4.3 Neighborhoods 34
4.3.1 Erase Function 34
4.3.2 Insert Function 35
4.3.3 Relocate Neighborhood 38
4.3.4 Swap Neighborhood 38
4.3.5 Setup Insertion and Setup Removal Neighborhoods 40
4.3.6 Infeasibility Strategy 41
4.4 Local Search 42
4.4.1 First Improvement 42
4.4.2 Random Variable Neighborhood Descent 43
4.5 Iterated Local Search 43
4.5.1 Perturbation 45
4.5.2 Acceptance Criteria 45

5 Computational Experiments **47**
5.1 Instances Description 47
5.2 Parameters 48
5.3 Results 48
5.4 Statistical Tests 53

6 Conclusions and Future Work **56**

A Detailed Results of the Methods for each Instance **61**

List of figures

Figure 1.1 The PLSV Connections Scheme 12
Figure 1.2 Manifold Connections 12
Figure 1.3 Water Injection Scheme 13
Figure 1.4 Production Wells and Water Injection Wells Pipelines 14

Figure 4.1 Solution Representation and Starting Times 27
Figure 4.2 Schedule Construction Procedure Scheme 29
Figure 4.3 First Allocation on Schedule Construction Procedure 30
Figure 4.4 Second Allocation on Schedule Construction Procedure 31
Figure 4.5 Third Allocation on Schedule Construction Procedure 31
Figure 4.6 Fourth Allocation on Schedule Construction Procedure 32
Figure 4.7 Fifth Allocation on Schedule Construction Procedure 32
Figure 4.8 Erase Function - Single Operation Batch 35
Figure 4.9 Erase Function - Multiple Operation Batch 35
Figure 4.10 Insert Function - case 1 36
Figure 4.11 Insert Function - case 2 36
Figure 4.12 Insert Function - case 3 37
Figure 4.13 Insert Function - case 4 37
Figure 4.14 Insert Function - case 5 37
Figure 4.15 Relocate Neighborhood - case 1 39
Figure 4.16 Relocate Neighborhood - case 2 39
Figure 4.17 Swap Neighborhood - case 1 40
Figure 4.18 Swap Neighborhood - case 2 40
Figure 4.19 Setup Insertion 41
Figure 4.20 Setup Removal 41

Figure 5.1 Boxplot - All Groups 52
Figure 5.2 Dominance Among Methods 53

List of tables

Table 2.1 Summary of Identical Parallel Machine Scheduling Studies 20

Table 4.1 Operations Data 29
Table 4.2 Machines Data 29

Table 5.1 Results for Group 1 49
Table 5.2 Results for Group 2 49
Table 5.3 Results for Group 3 50
Table 5.4 Results for Group 4 50
Table 5.5 Results for Group 5 51
Table 5.6 Results for Group 6 51
Table 5.7 ANOVA Tests Within All Pairs of Methods 54

Table A.1 Complete Results for each Method and Instance 61

1
Introduction

Pipe Laying Support Vessels (PLSV) are one of the most critical agents on the oil offshore exploration and production (E&P). These vessels are responsible for connecting sub-sea oil wells to the production platforms. Their high operational cost is reflected in expensive daily fleet rates Bremenkamp (2017). Efficiently planning these connections is a crucial step on the oil & gas supply chain management, once this is the last stage before a well can start the oil production. The exploration companies have an available fleet of these vessels to coordinate and need to define a schedule for each PLSV to perform interconnection operations at a set of oil wells located off-shore. Any improvement on this schedule is reverted in a great amount of money saved, which can be measured by oil production, fleet rental cost, tardiness in the whole operation and other key factors. One or more of these objectives must be considered for the company to optimize their schedule providing a relevant competitive advantage.

The PLSV operation starts at the port, with the loading of the necessary equipment to perform the connection procedures. Once the vessel is loaded, it navigates to reach the off-shore wells where the operations are performed. The PLSV connects the lines between the platform and the wells and then returns to the port to reload and repeat this process to perform more operations. This whole cycle is planned for a one-year time horizon in most cases. An illustration of the platform, lines and wells is shown in Figure 1.1

There are three main families of operations that the PLSV perform: wells interconnections, which are the most relevant and common activities; manifold installations, allowing a single production platform to be connected with many oil wells; and scheduled vessel maintenance, which are related to the vessel and not the wells but is considered as an operation in the schedule. An image of a manifold installation is exposed in Figure 1.2.

Regarding the vessels, each one has a specific release date, due to contract characteristics. PLSV vessels can be eligible or not to perform a specific operation since their fleet is heterogeneous. Another aspect to be considered is the occupancy capacity, rated on the vessel's deck that will be used to store the necessary equipment.

Figure 1.1: The PLSV Connections Scheme

Source: CENPES/Petrobras

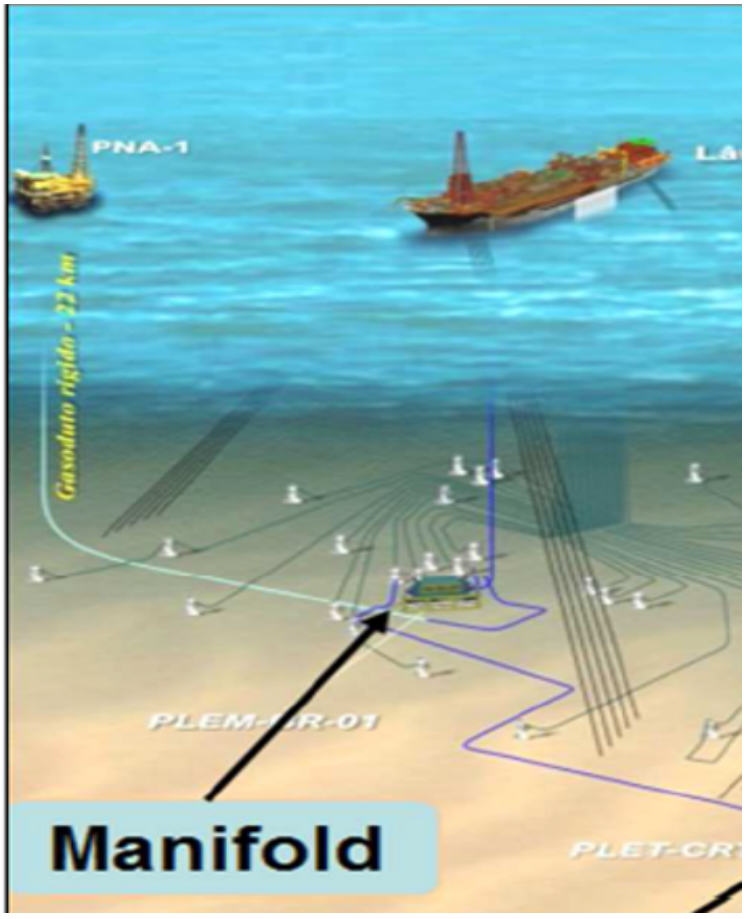

Figure 1.2: Manifold Connections

Source: CENPES/Petrobras

In Figure 1.3, a water injection well illustration is shown. The oil production wells can be rising wells, that have enough pressure to elevate the oil from the reservoir, and the non-rising wells, that need lifting methods such as water and gas injection to pressurize the well and elevate the oil.

Regarding the petroleum production, there are two types of wells that the PLSV operate on: the oil production wells, which are responsible for the oil elevation from the reservoir to the platform and the water/gas injection wells, that energize the reservoir, guaranteeing a constant pressure after the oil removal.

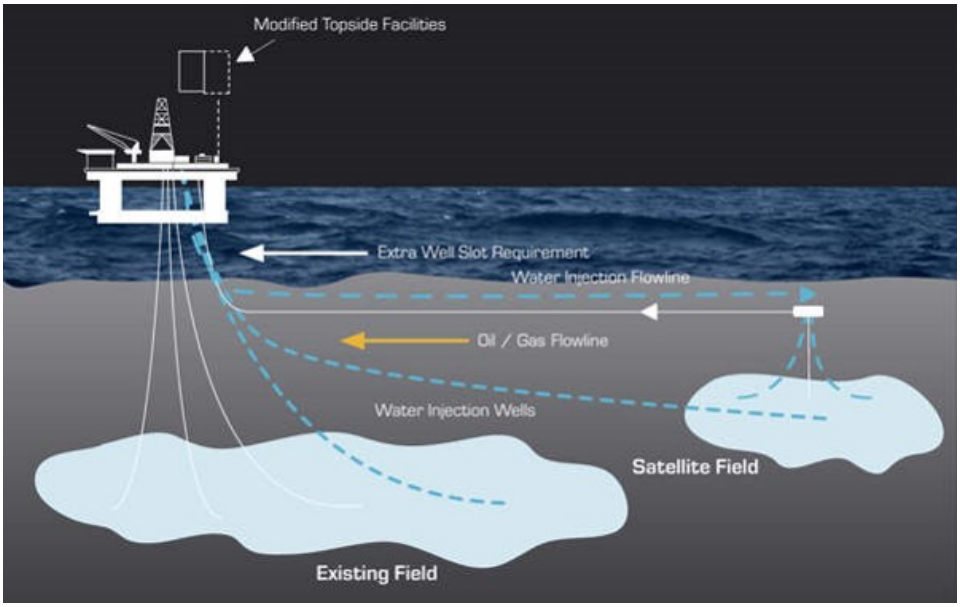

Figure 1.3: Water Injection Scheme

Source: Offshore Technology

For the wells to be considered complete, different kinds of operations need to be performed. These operations have release dates, related to the arrival of the necessary equipment, and estimated processing times. The production rates in each well varies and there is dependence between them, where a well needs others to be finished in order to start the oil production. The oil production wells have three kinds of line that the PLSV will connect to the platform: the production line, which will transport the petroleum from the reservoir to the platform, the umbilical line that opens and shuts the wells valves, and the annular lines that are used to pressurize the wells and elevate the oil. The water and gas injection wells require two kinds of pipelines: the annular line and the umbilical line. These lines are explained in Figure 1.4.

The PLSV is an extremely limited and expensive resource, influencing decision makers to search for the available tools to optimize their activities. The problem studied in this work consists on delivering a schedule plan that determines: which operations will be performed by each vessel, the date and order of these operations and how this schedule will be divided in PLSV voyages. The structure of these voyages consists on a setup time, that considers the loading time of the necessary equipment and navigation time to the wells,

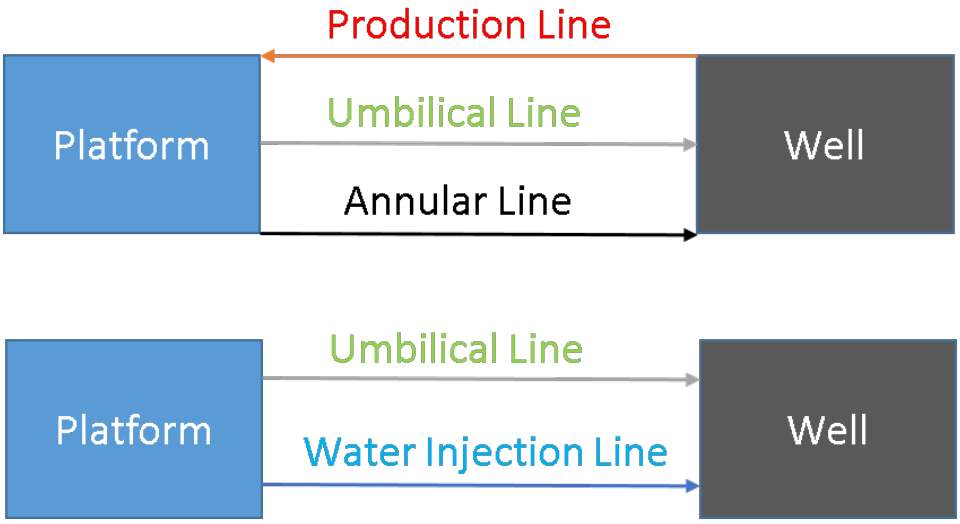

Figure 1.4: Production Wells and Water Injection Wells Pipelines

Source: The Author

followed by one or more operations of the same family to be performed. These setup times are classified as non-anticipatory because they can only start when all operations of the voyage are released (due to equipment availability). The objective is finishing the most productive wells as soon as possible, maximizing the oil production.

Due to the characteristics of the PLSV scheduling problem, we can model it as an identical parallel machine scheduling problem. The vessels are interpreted as machines that perform operations. Each operation is associated to a well, which we will call a job, following the common parallel machine scheduling problem notation. In that notation, the operations are the elements that are processed by the machines, while the jobs are associated to a set of operations. In that way, the jobs are not processed directly by machines, but when all operations associated to a specific job are finished, it is considered complete. The family of operations is related to their types and for each family we have a different setup time associated. Another important aspect of the machine scheduling problem is the association between the batches and PLSV voyages. The batches are formed by one setup time followed by a sequence of operations. In the PLSV problem any voyage starts with loading the vessels and navigating to the wells (analogous to the machine scheduling setup times) to perform a sequence of operations.

Identical parallel machine scheduling has been intensively studied in the past 20 years, in many different applications and approaches. The objective of this work is to develop efficient algorithms to minimize the weighted completion time on identical parallel machine scheduling problems with non-anticipatory

family setup times, eligibility between machines and operations, release dates of operations and machines, job-operations associations and non-preemptive operations. The secondary objective of this work is to solve the real-life application that motivated this study which is the PLSV scheduling problem.

The main contributions of this work is providing a local search procedure as well as a metaheuristic algorithm that reach high quality solutions for the described problem in competitive time. Another important contribution is running extent computational experiments in a large set of instances based in real PLSV problem data.

The remainder of this work is divided as follows. In Chapter 2, a complete literature review on identical machine scheduling problem is presented. In Chapter 3, we describe the problem and present a mathematical formulation in detail. In Chapter 4, the methodology of the work is introduced, including the implemented constructive heuristic, explored neighborhood structures, the local search procedure and the Iterated Local Search metaheuristic. In Chapter 5, computational experiments are shown with structured results and in Chapter 6, we present the conclusions of the thesis and recommended future work.

2
Literature Review

The parallel machine scheduling problem with setup times has been intensively studied in the literature for the past 20 years. In this study we focus only on the identical parallel machine scheduling problem with setup times, but there are several other considerations on non-identical parallel machine problems, with and without setup times and other modelling approaches. Potts and Kovalyov (2000) provided a extensive literature review on scheduling with batching, not only in parallel machines, explaining the basic algorithms and focused on dynamic programming methods to solve this kind of problems. These algorithms consist in creating a forward recursion appending jobs in a specific schedule and a backward recursion inserting jobs in the beginning of a schedule. We suggest this article to understand problems that treat non-identical parallel machine scheduling.

We selected 21 of the most relevant papers aimed to solve identical parallel machine scheduling problems with family setup times. The solution approaches presented vary from mixed-integer linear problem formulations to heuristics and metaheuristics. Some of the articles are focused on real life problems applications while most of them are based on theoretical problems. We limited our research to articles published in the last twenty years, written in English, related to identical parallel machine scheduling problems and published in journals or book series. The search was performed on the Scopus platform and the key words searched were: "identical parallel machine scheduling problem" and "batch scheduling problem" A summary of all studied publications is shown in Table 2.1.

Regarding exact methods, Webster and Azizoglu (2001) addresses the problem on scheduling identical parallel machines with family setup times in order to minimize total weighted flowtime. They also present two dynamic programming algorithms, named backward and forwards procedures. They show that when the number of machines and families are fixed, the backward recursion is executed in polynomial time in the sum of the weights and the forward recursion is executed in polynomial time in the sum of times (both setup and processing). Comparing then the sum of weights and times will define which

method will be more effective for each problem. Chen and Powell (2003) propose a column generation and branch-and-bound techniques to solve both sequence-dependent and independent problems on identical parallel machines. These techniques consist in solving linear relaxations of the problem by standard column generation procedures. The focus of the method is based on node selection and branching variable selection to effectively solve these relaxations. They aim to solve both minimizing total weighted completion time problems and minimizing tardiness. Azizoglu and Webster (2003) also propose branch-and-bound algorithms to solve identical parallel machine scheduling problems with family setup times, but the objective function is to minimize total weighted flowtime. The authors take in consideration some dominance properties based on the shortest weighted processing time (SWPT) order and lower bound definitions to propose an efficient branch-and-bound method. Dunstall and Wirth (2005b) improve the branch-and-bound methods aiming to minimize the weighted completion time by proposing a different branching scheme able to shorten the search tree size. They propose two branch-and-bound algorithms that use different branching schemes. They also use techniques to eliminate redundancies on solutions and consider upper bounds by scheduling entire families into machines with a SWPT rule. Dunstall and Wirth (2005a) propose a method to solve the problem with sequence-dependent setup times. They develop a four stage algorithm by mustering all jobs of a family in a single batch and allocating to machines, re-sequencing the batches at single machines, splitting the batches and at last performing a single machine sequencing phase. These steps are based on dominance rules between jobs and families. Omar and Teo (2006) develop a mixed-integer programming approach to minimize both earliness or tardiness on identical parallel machine scheduling with sequence-dependent setup times. Basically, they improve a formulation by adding a constraint ensuring that a job and its successor will be produced in the same machine.

Regarding heuristics and metaheuristics applied to similar problems, Min and Cheng (1999) propose a genetic algorithm based on machine code to minimize the makespan in identical parallel machine scheduling problems. In the coding phase the gene code is also the number of the machine which the job is processed, reason why the method is named genetic algorithm based on machine code. After the genetic algorithm, the authors develop a simulated annealing metaheuristic. Mendes et al. (2002) compare the performance of two different metaheuristic methods to minimize makespan in identical parallel machine scheduling with family setup times. The first metaheuristic explored is a tabu search with four different types of insertion/removal moves. The second

is a memetic approach combining a population-based method with local search procedures. Ruiz and Maroto (2005) work consists on reviewing and comparing heuristics and metaheuristics for the permutation flowshop problem to minimize makespan. In the paper, they approach 25 different methods, varying from the most antique and classic heuristics to some recent metaheuristics, including genetic algorithms, tabu searches, iterated local searches and hybrid techniques. Liao et al. (2012) studied the identical parallel machine scheduling problem with setup times focused on minimizing the weighted completion time. The authors propose an heuristic that aims to decrease the setup times and balance the weighted completion time on each machine. There are two steps involved: the first is to use the a tabu search to schedule jobs in each machine. The second contrasts the obtained schedule with another without setup times by changing the assignment of jobs to machines. These exchanges are subject to certain conditions based on dominance rules such as the SWPT Weighted Mean Processing Time. Schaller (2014) presents a set of procedures for scheduling identical parallel machines with family setup times focused on minimizing total tardiness. They test these procedures in a large set of instances and conclude that for this specific problem, the genetic algorithms are usually more effective. Mehdizadeh et al. (2015) develop a metaheuristic algorithm called vibration damping optimization (VDO), based on mechanical vibration, to solve the identical parallel machine scheduling problem with sequence-independent setup times. The VDO algorithm consists in iteratively and randomly changing the current solution based on oscillating the solution around an amplitude parameter. Van Der Zee (2015) proposes a simulation study on family-based dispatching heuristics to identical machine scheduling. They test and compare existing heuristics for the problem and create new heuristics by extending the existing ones, simulating the results on a large set of instances. Wu et al. (2018) introduced a modified Water Flow-Like Algorithm (WFA) to solve the identical parallel machine scheduling problem with sequence-dependent setup times, considering due dates, order profit and tardiness penalties. Their strategy was to develop a procedure similar to the variable neighborhood search and mix it with the WFA, by splitting and moving operations. Dipak and Gupta (2018) presented an improved cuckoo search algorithm (ICSA) focused on minimizing makespan on identical parallel machine scheduling. Their first step consists on generating a set of initial population schedules, combining the longest processing time rule with job-interchange procedures, and selecting the best schedule. The authors then propose a heuristic approach to transform continuous positions in the CSA into discrete job schedules. At last, an heuristic algorithm related to pairwise exchange neighborhood is introduced to complete the ICSA procedure.

Nguyen et al. (2018) propose an heuristic of layering and adapting (HLA) to minimize both energy consumption and makespan in identical parallel machine scheduling problems. The algorithm strategy is to consider resource allocations of jobs as flexible form articles to be packed in a strip with variant heights throughout its length. Two local search procedures are used with the concepts of adapting process and greedy repackage process.

Considering different real-life applications of the identical parallel machine scheduling problem, Shin and Leon (2004) applies a scheduling approach to module processing in thin film transistor liquid crystal display manufacturing. The authors develop a multifit heuristic and a tabu search to solve the problem focused on minimizing total tardiness and minimizing total family setups. Ciavotta et al. (2016) propose dynamic heuristics and candidates reduction strategies with the multiheuristic rollout procedures, applying them in illustrative case studies. The case study was first proposed by Pacciarelli et al. (2011) and is based on a pharmaceutical manufacturing plant. Kaplan and Rabadi (2011) studies the aerial refuelling scheduling problem with release dates, and due date deadline windows, focused on minimizing the total weighted tardiness. A simulated annealing metaheuristic was proposed as well as a metaheuristic with randomized priority search, using an apparent priceswise tardiness cost with ready times as a dispatching rule. Lee (2017) studies a real life production scheduling from a plant of Acrylonitrile-Butadiene-Styrene plate products. The focus is to minimize tardiness and the strategy developed proposes a dispatching rule to provide an initial solution and then applying an iterated greedy search metaheuristic to improve the results. Abu-Marrul et al. (2020a)studies the PLSV scheduling problem. Three mixed-integer linear formulations are proposed to solve the problem aiming to minimize completion time: the positional, the batch, and the time-index models. Abu-Marrul et al. (2020b) also studies the PLSV scheduling problem. A constructive heuristic is presented to minimize the weighted completion time.

Table 2.1: Summary of Identical Parallel Machine Scheduling Studies

Work	**Year**	**Approach**	**Main Method**
Potts and Kovalyov (2000)	2000	Literature Review	Dynamic Programming
Webster and Azizoglu (2001)	2001	Exact Methods	Dynamic Programming
Chen and Powell (2003)	2003	Exact Methods	Column Generation
Azizoglu and Webster (2003)	2003	Exact Methods	Branch & Bound
Dunstall and Wirth (2005b)	2005	Exact Methods	Branch & Bound
Dunstall and Wirth (2005a)	2005	Exact Methods	Branch & Bound
Omar and Teo (2006)	2006	Exact Methods	Mixed Integer Programming
Min and Cheng (1999)	1999	Heuristics and Metaheuristics	Genetic Algorithm
Mendes et al. (2002)	2002	Heuristics and Metaheuristics	Tabu Search
Ruiz and Maroto (2005)	2005	Heuristics and Metaheuristics	Genetic Algorithm
Liao et al. (2012)	2012	Heuristics and Metaheuristics	Crauwels Tabu Search
Schaller (2014)	2014	Heuristics and Metaheuristics	Genetic Algorithm
Mehdizadeh et al. (2015)	2015	Heuristics and Metaheuristics	VDO Algorithm
Van Der Zee (2015)	2015	Heuristics and Metaheuristics	Simulation Model
Wu et al. (2018)	2018	Heuristics and Metaheuristics	Water Flow Like Algorithm
Dipak and Gupta (2018)	2018	Heuristics and Metaheuristics	Improved Cuckoo Search Algorithm
Nguyen et al. (2018)	2018	Heuristics and Metaheuristics	Heuristic of Layering and Adapting
Shin and Leon (2004)	2004	Real-Life Application	Tabu Search
Kaplan and Rabadi (2011)	2011	Real-Life Application	Simulated Annealing
Ciavotta et al. (2016)	2016	Real-Life Application	Multiheuristic Rollout Procedure
Lee (2017)	2017	Real-Life Application	Iterated Greedy Search
Abu-Marrul et al. (2020a)	2019	Real-Life Application (PLSV)	Mixed Integer Programming
Abu-Marrul et al. (2020b)	2020	Real-Life Application (PLSV)	Constructive Heuristic

3
Mathematical Formulation

3.1
Problem Description

As mentioned in Section 1, the PLSV scheduling problem can be modeled as an identical parallel machine scheduling problem, due to the similar characteristics between them. The vessels are analogous to the machines, which will process the operations in a specific well, which we call job. The operations are processed by the machines and are associated to jobs, but the jobs are not directly processed. When all operations associated to a job is finished, the job is considered complete. In this work we will use machine scheduling notation to describe the problem and the further mathematical formulation.

Let $\mathcal{O}$ be the set of operations, each one belonging to a family of the set $\mathcal{F}$. These operations are scheduled in a set $\mathcal{M}$ of machines, which will process these operations in a particular order. Associated to each family, there is a setup time s_f, independent from the machine and considered whenever there is a change on the family of operations or when the machine capacity q_k is reached. The setup time s_f marks the beginning of a voyage, which in machine schedule notation is called a batch. The problem studied is classified as non-preemptive, meaning that operations must be processed at once and cannot be divided. Regarding the operations $i \in \mathcal{O}$, we consider their processing times p_i and release dates r_i. All operations must be associated to at least one job, defined in the set $\mathcal{N}$, meaning that jobs are affected by this operation. One operation can be related to more than one job, indicating the dependence between these jobs. The subset of operations that compose a job j is defined as $\mathcal{O}_j$. These jobs have weights w_j related to their production rate. To consider a job completed all the associated operations must be finished. On the other way, the subset $\mathcal{N}_i$ defines the jobs related to an operation i. Each machine has a release date r_k that means the date when it is available to operate. We also consider eligibility between machines and operations, denoted by the subset M_i indicating which machines can process each operation i.

The vessels voyages are considered as batches on the parallel machine

scheduling problem. They consist on a sequence of operations of the same family preceded by a setup time. This setup time represents the loading of the necessary equipment on the origin port and the navigation time to reach the wells. Due to that, the setup times can only start when all operations of the same batch are released (meaning that all necessary equipment for performing every operation of the voyage has arrived). This particularity is called non-anticipatory setup time. These batches must respect the capacity of the machine q_k, by summing the load occupation l_i of the operations scheduled on this batch.

The objective function of the problem is to minimize the weighted completion time of all jobs, $\sum_{j \in \mathcal{N}} w_j C_j$, subject to all mentioned constraints. The completion time of each job, C_j, is calculated by the maximum completion time of all associated operations: $\max_{i \in \mathcal{O}_j} C_i$. The objective is to finish as soon as possible the most productive jobs.

Regarding the complexity of the PLSV scheduling problem, Mokotoff (2004) prove that the classical identical parallel machine scheduling problem is `NP-hard`. By simplifying some parameters of the PLSV scheduling problem, we can model it exactly as the classical identical parallel machine scheduling problem. To do that we must set all operations to be eligible to all machines, all operations being associated with a single job, set the families of all operations as 1 and the correspondent setup times as zero and setting the release dates of all operations and machines as the first day of the time horizon. Since we can model the PLSV scheduling problem as the classical identical parallel machine scheduling problem, which we know is `NP-hard`, and the complete PLSV scheduling problem has even more constraints and parameters, we can conclude that the PLSV scheduling problem is also `NP-hard`.

3.2 Positional Scheduling Formulation

In this section we present a positional scheduling formulation described by Unlu and Mason (2010) and extended by Abu-Marrul et al. (2020a) for the PLSV problem. There are other possible formulations for this problem, such as the Time-index, the Bucket-index and the Batch Scheduling formulation. We introduce this positional schedule formulation because it fits well the solution representation presented in this study methodology, presented in the next chapter, where the machines are vectors that contains an element (setup or operation) in each position. The positional formulation illustrates the problem's constraints and parameters and inspired the proposed metaheuristic presented in this study.

The positional scheduling formulation consists on representing the machines with a sequence of positions. The model then defines, for each position, if it will allocate an operation or a setup time, respecting all constraints. In our case, the resulting schedule aims to minimize the weighted completion time of the jobs, but it can optimize a different objective function, such as minimize tardiness, maximize production, minimize completion time, among others.

The set $\mathcal{P}_k$ indexes the positions of each machine. The decision of allocating an operation or a family setup time is represented by two binary variables: X_{ik}^p representing an operation i allocated at the p-th operation of the machine k, and Y_{fk}^p representing a setup time on the p-th position of machine k of the family f.

To keep track of some critical information during the model's decisions the authors define three other continuous variables, S_k^p defines the start of each position p in machine k and C_i and C_j, representing the completion time of operations and jobs, respectively.

To deal with the occupation constraint, the continuous variables L_k^p are added representing the cumulative load occupancy in position p. To treat the non-anticipatory setup times, the variables R_k^p represent the release time of the position p, to guarantee that the start of positions which have setup times allocated will respect the release dates of all operations on the batch.

The positional scheduling formulation is then described bellow:

$$\min \sum_{j \in \mathcal{N}} w_j C_j \tag{3-1}$$

subject to

$$\sum_{k \in \mathcal{M}_i} \sum_{p \in \mathcal{P}_k} X_{ik}^p = 1 \qquad \forall i \in \mathcal{O} \qquad (3\text{-}2)$$

$$\sum_{i \in \mathcal{N}} X_{ik}^p + \sum_{f \in \mathcal{F}} Y_{fk}^p \leq 1 \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}3)$$

$$L_k^p \geq L_k^{(p-1)} + \sum_{i \in \mathcal{O}} l_i X_{ik}^p - \sum_{f \in \mathcal{F}} q_k Y_{fk}^p \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}4)$$

$$L_k^p \leq q_k \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}5)$$

$$\sum_{i \in \mathcal{N}_f} X_{ik}^p \leq \sum_{i \in \mathcal{N}_f} X_{ik}^{(p-1)} + Y_{fk}^{(p-1)} \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k, f \in \mathcal{F} \qquad (3\text{-}6)$$

$$R_k^p \geq \sum_{i \in \mathcal{O}} r_i X_{ik}^{(p+1)} \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}7)$$

$$R_k^p \geq R_k^{(p+1)} - \sum_{f \in \mathcal{F}} r_{max} Y_{fk}^{(p+1)} \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}8)$$

$$S_k^p \geq S_k^{(p-1)} + \sum_{i \in \mathcal{O}} p_i X_{ik}^{(p-1)} + \sum_{f \in \mathcal{F}} s_f Y_{fk}^{(p-1)} \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}9)$$

$$S_k^p \geq r_k \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}10)$$

$$S_k^p \geq R_k^p \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}11)$$

$$C_i \geq S_k^p + p_i - (1 - X_{ik}^p)M \qquad \forall i \in \mathcal{O}, k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}12)$$

$$C_j \geq C_i \qquad \forall j \in \mathcal{N}, i \in \mathcal{O}_j \qquad (3\text{-}13)$$

$$X_{ik}^p \in \{0, 1\} \qquad \forall i \in \mathcal{O}, k \in \mathcal{M}_i, p \in \mathcal{P}_k \qquad (3\text{-}14)$$

$$Y_{fk}^p \in \{0, 1\} \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k, f \in \mathcal{F} \qquad (3\text{-}15)$$

$$C_i \geq 0 \qquad \forall i \in \mathcal{O} \qquad (3\text{-}16)$$

$$L_k^p, S_k^p, P_k^p \geq 0 \qquad \forall k \in \mathcal{M}, p \in \mathcal{P}_k \qquad (3\text{-}17)$$

The Objective Function (3-1) is to minimize the weighted completion time of the jobs. Constraints (3-2) guarantee that every operation will be allocated once in the schedule. Constraints (3-3) ensure that we schedule a setup time or an operation in each position, not both. Constraints (3-4) keep track of the occupation load in each position, summing the occupancy load when an operation is allocated in the position and setting the variable to zero when it is a setup. Constraints (3-5) ensure that the machine occupancy capacity is respected. Constraints (3-6) forbid an operation of one family to be scheduled after an operation or setup time of a different family. Constraints (3-7) calculate the release of a position based on the release date of the next operation allocated on that machine. Constraints (3-8) guarantee that the release of a position is greater or equal than the release variable of the next position. These constraints (3-7) and (3-8) together ensure that the non-anticipatory setup times will be respected. Constraints (3-9) compute the start time of a position

by summing the start time of the precedent position with the processing time of the scheduled operation or setup time allocated. Constraints (3-10) forbid the start time of a position in a machine to be lower than the release date of the machine. Constraints (3-11) guarantee that a position can only start after it is released. Constraints (3-12) compute the completion time of operations by summing the start time of the position with the processing time of the allocated operation. Constraints (3-13) force the completion time of a job to be the last completion time of all operations associated to that job. Constraints (3-14)-(3-17) are the non-negativity and domain constraints.

4
Methodology

The first aspect of this work methodology is the solution representation. Based on it, the methods are developed aiming to minimize the weighted completion time of jobs. The next step of the methodology is obtaining an initial solution for the identical parallel machine scheduling problem using a constructive heuristic. After that, we define the neighborhoods that will be explored in our search: Relocate, Swap, Setup Insertion and Setup Removal. With these neighborhoods defined, we propose a Local Search procedure. At last, we present the Iterated Local Search metaheuristic that looks for high quality solutions for the PLSV Scheduling Problem. The next sections describe in detail each one of these steps.

4.1
Solution Representation And Evaluation

The positional formulation introduced inspired our solution representation for this work. In the mathematical formulation, the machines can be represented as vectors. For each machine, there are n positions containing an operation or a setup time. Representing the solutions as vectors of integers saves computational time when performing movements within the solution, but neglects the starting time information for each operation and setup time, which are used to calculate our objective function of minimizing the weighted completion time. To deal with this problem, we created an evaluation function that receives the vectors of integers and calculates the minimum starting time for each position respecting all mentioned constraints. The evaluation function runs in linear time in the number of operations. In Figure 4.1, the solution representation is exposed and then the solution representation indicating the starting times for each element.

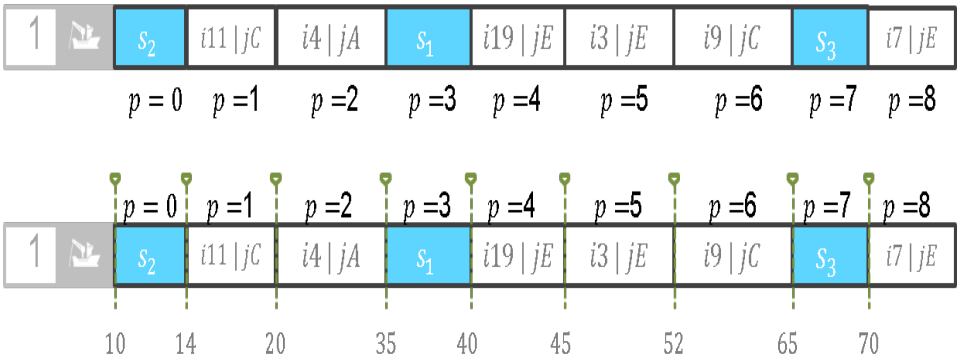

Figure 4.1: Solution Representation and Starting Times

4.2 Constructive Heuristic

In this section, we present the Schedule Construction Procedure, proposed by Abu-Marrul et al. (2020b), to provide an initial solution for our problem. Since it is an important part of this study methodology we will explain in detail the approach.

The Schedule Construction Procedure decides at each iteration, for an operation, in which machine it will be allocated and if it is allocated in an existing batch or a new one. The authors define a set that at first contains all operations. To define the order of operations to be allocated, a set of dispatching rules to rank all operations is considered. These dispatching rules will be explained further in subsection 4.2.1. If two operations are tied on the dispatching rule criteria, the algorithm chooses the one with the lower index. To decide in which machine the operation is allocated, the machines' completion times are evaluated. The operation will be performed by the machine that has the lower machine completion time including the recently allocated operation, respecting the eligibility constraints and the machines release dates. To decide if the operation will enter in an existing batch or a new one, the family and capacity constraints are checked. After the operation is allocated, the method removes it from the set of unscheduled operations. The algorithm runs until all operations are scheduled. It is summarized in Algorithm 1.

Algorithm 1 Schedule Construction Procedure

$C_k \leftarrow r_k, S_k \leftarrow r_k,\ L_k \leftarrow 0,\ F_k \leftarrow 0,\ \mathcal{B}_k \leftarrow \emptyset, \sigma_k \leftarrow \emptyset :\ \forall k \in \mathcal{M}$
$C_i \leftarrow \infty :\ \forall i \in \mathcal{O}$
$\mathcal{U} \leftarrow \mathcal{O}$
while there exists operations not assigned **do**
    Select operation $i^* \in \mathcal{U}$ and machine $k^* \in \mathcal{M}_{i^*}$ according to a chosen heuristic, defining *same* as true or false
    if *same* = true **then**
        $S_{k^*} \leftarrow \max(r_{i^*}, S_{k^*}),\ C_{k^*} \leftarrow C_{k^*} + \max(0, r_{i^*} - S_{k^*}) + p_{i^*}$
        $L_{k^*} \leftarrow L_{k^*} + l_{i^*},\ \mathcal{B}_{k^*} \leftarrow \mathcal{B}_{k^*} \cup \{i^*\}$
    else
        $S_{k^*} \leftarrow \max(r_{i^*}, C_{k^*}),\ C_{k^*} \leftarrow \max(r_{i^*}, C_{k^*}) + s_{f_{i^*}} + p_{i^*}$
        $L_{k^*} \leftarrow l_{i^*},\ \mathcal{B}_{k^*} \leftarrow \{i^*\}$
        $\sigma_{k^*} \leftarrow \sigma_{k^*} \cup \{f_{i^*}\}$
    end if
    $\sigma_{k^*} \leftarrow \sigma_{k^*} \cup \{i^*\},\ F_{k^*} \leftarrow f_{i^*},\ C_{i^*} \leftarrow C_{k^*},\ \mathcal{U} \leftarrow \mathcal{U} \setminus \{i^*\}$
end while
return σ

The Schedule Construction Procedure first sets the initial values for the variables: completion time of each machine, C_k, and starting time for the current batch for each machine, S_k, as the release date of that machine, r_k. The loading occupancy of the current batch for each machine, L_k, the family of the current batch in the machine, F_k, the set of operations scheduled in the current batch in the machine, B_k, and the list of schedules for each machine, σ_k, are set as zero. While there are unscheduled operations, the method selects one operation and machine, based on a specific dispatching rule. Then the algorithm checks if it is feasible to schedule it in an existing batch. If it is, the operation is inserted in this batch, the variables that control the starting time of the batch in the machine, completion time of the machine, load occupancy capacity of the machine, and operations scheduled in the batch in the machine are updated. Otherwise, the operation is scheduled in a new batch, including it's respective setup time. The variables that control the starting time of the batch in the machine, completion time of the machine, load occupancy of the batch in the machine, operations scheduled in the batch in the machine and list of schedules in the machine are updated. The algorithm stops when all operations are scheduled.

An illustrative scheme of the Schedule Construction Procedure is exposed in Figure 4.2. We will illustrate the Schedule Construction Procedure in a small example, using the Earliest Release Date (ERD) dispatching rule to explain how the method construct batches. The ERD rule consists on ordering all operations by the release date criteria. The operations that have earlier release

dates are scheduled first. The operations and machines data are presented in Tables 4.1 and 4.2 ordered by their respective release dates.

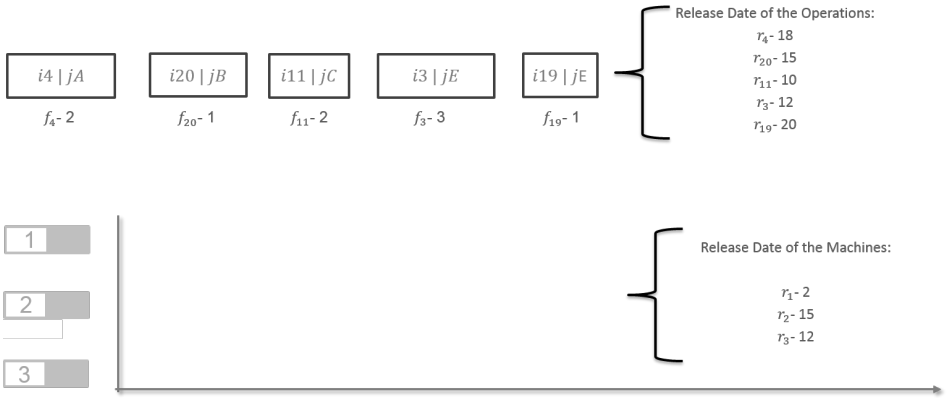

Figure 4.2: Schedule Construction Procedure Scheme

Source: The Author

Table 4.1: Operations Data

$\mathcal{O}$	**Family**	**Release Date**
i_{11}	2	10
i_3	3	12
i_{20}	1	15
i_4	2	18
i_{19}	1	20

Table 4.2: Machines Data

$\mathcal{M}$	**Release Date**
k_1	2
k_3	12
k_2	15

In this illustrative example, we have five operations, belonging to four jobs and three different families. For example $i_4|jA$ means that operation i_4 is associated to the job jA and the family of an operation is expressed as f_i. Each of these operations and machines have a specific release date. To simplify, in this example we consider that all operations are eligible at all machines and we do not consider occupation constraints, but in the real problem it is not possible to allocate operations in machines that are not eligible to perform them or create batches that surpasses the machine capacity. The algorithm starts by selecting the first operation to be allocated. To do that the method uses one of the dispatching rules proposed. In this illustrative example, we are using the ERD dispatching rule to decide in each order the operations will be

allocated. In spite of not being the best dispatching rule in terms of results, we decided to use it in the example because it is intuitive to understand how it ranks the operations, by simply selecting the operations with lower release dates first. The Schedule Construction Procedure is exactly the same regardless of the dispatching rule used to prioritize the operations, the only difference between them is the order of allocation. Once the operation is chosen, the algorithm allocates it in the machine that will give the minimum completion time. The procedure selects the operation i_{11}, which is the first to be released at day 10 and allocate it at machine 1 because it is released on day 2, as the other machines are not released yet in day 10. The first allocation is presented in Figure 4.3.

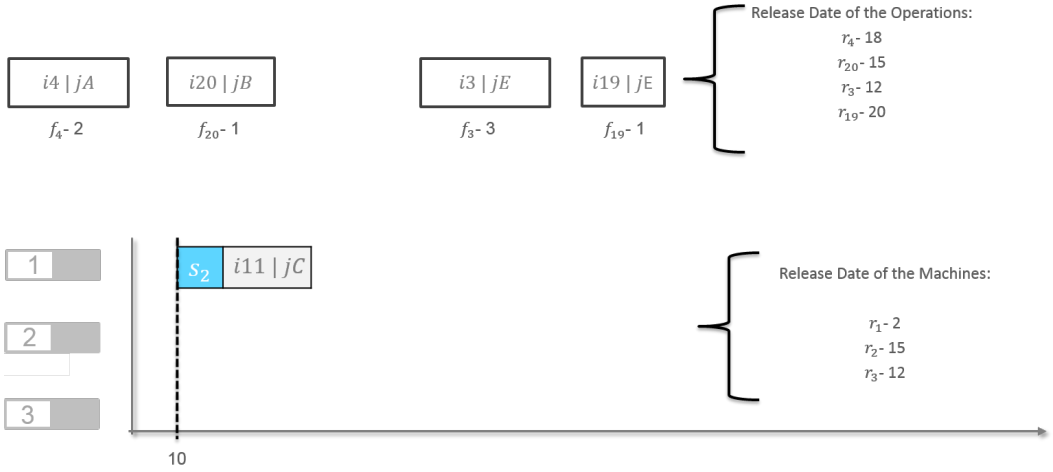

Figure 4.3: First Allocation on Schedule Construction Procedure

Source: The Author

Following the ERD rule, the second operation to be scheduled is the operation i_3, which is released on day 12. It can be allocated in machine 1 creating a new batch after the end of operation i_{11}, because it is from a different family, in machine 2 starting on day 15, when the machine will be released, or in machine 3 starting on day 12. As the algorithm aims to allocate it in the machine that will have the lowest completion time, it is allocated in machine 3, as illustrated in Figure 4.4.

The next operation to be allocated following the ERD rule is i_{20}. It can be scheduled in machine 1, but since it is from a different family than i_{11}, it will create a new batch after it. The same will happen in machine 3, where it will create a new batch after operation i_{13}. Since machine 2 is released on day 15 we can allocate operation i_{20} and have the minimum machine completion time, as we can see in Figure 4.5.

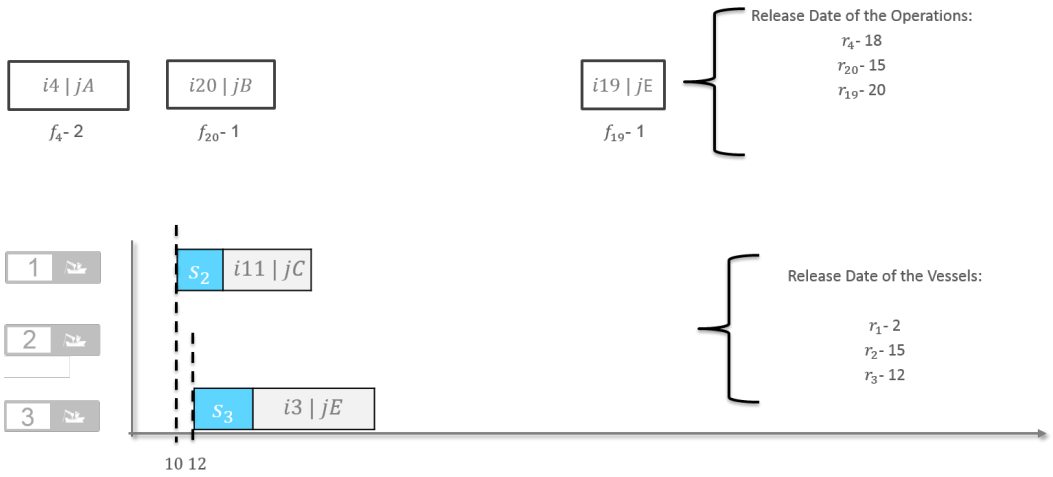

Figure 4.4: Second Allocation on Schedule Construction Procedure

Source: The Author

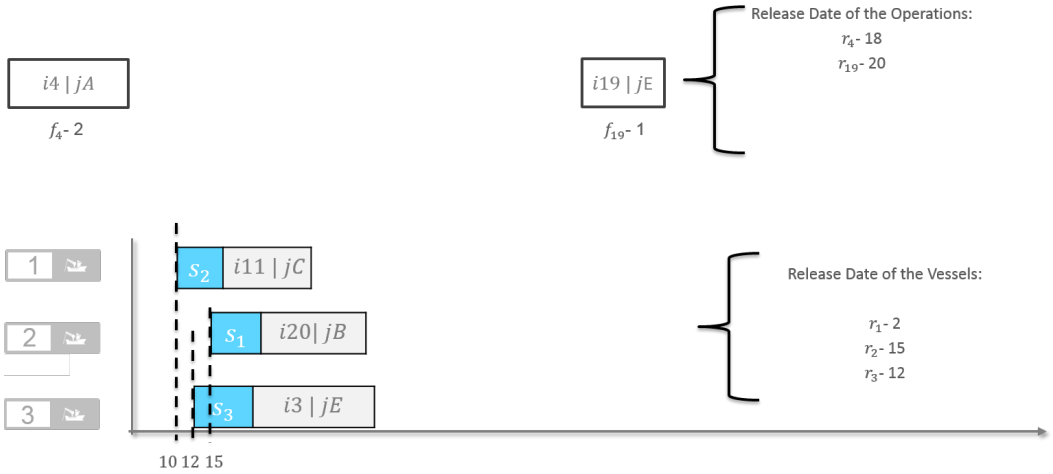

Figure 4.5: Third Allocation on Schedule Construction Procedure

Source: The Author

Observing the release date of the remaining operations, we notice that the next one to be scheduled is operation i_4. This operation is from family 2, so it can be allocated in machine 2 or 3, creating new batches, or in machine 1 in the same batch of operation i_{11}. One important characteristic of this step is that by adding this operation to the same batch on machine 1 we need to shift all batch forward due to the non-anticipatory setup times. Evaluating the completion time of machines, we see that this is the best allocation in this scenario, moving the setup of this batch from day 10 to day 18, as observed in Figure 4.6.

To allocate the last operation, the same logic is applied. The operation i_{19} is from family 1, the same of operation i_{20}. So it can be allocated in the same batch on machine 2 or in new batches on machines 1 and 3. Scheduling it in

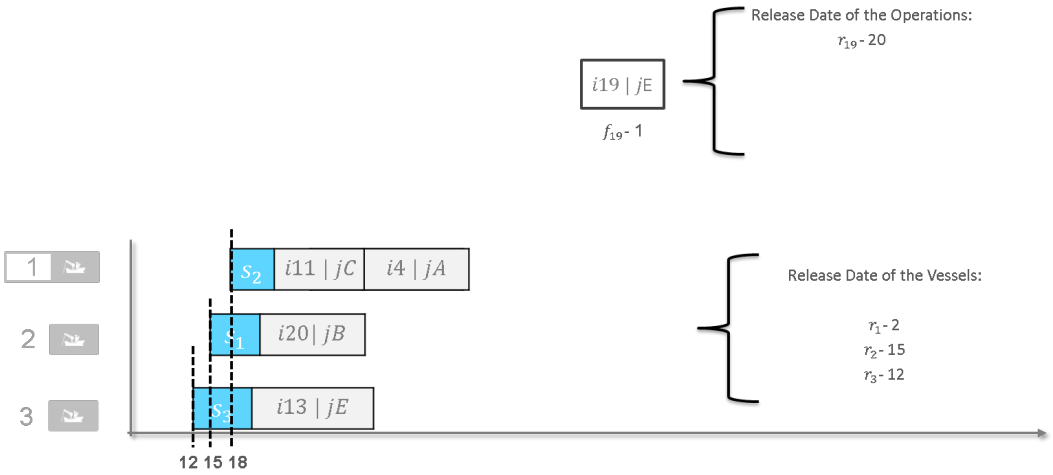

Figure 4.6: Fourth Allocation on Schedule Construction Procedure

Source: The Author

machine 2 will shift the whole batch to start at the release date of operation i_{19}, day 20, because of the non-anticipatory setup times. Considering the machine completion time, this is the best way to allocate operation i_{19}, represented on Figure 4.7. Since all operations are allocated, we can consider that we have an initial solution for this illustrative problem. To evaluate this solution we calculate the weighted completion time of jobs as described on Section 3.1. This procedure is taken aiming to provide an initial solution as input to the local search procedures and metaheuristic that will follow on the study.

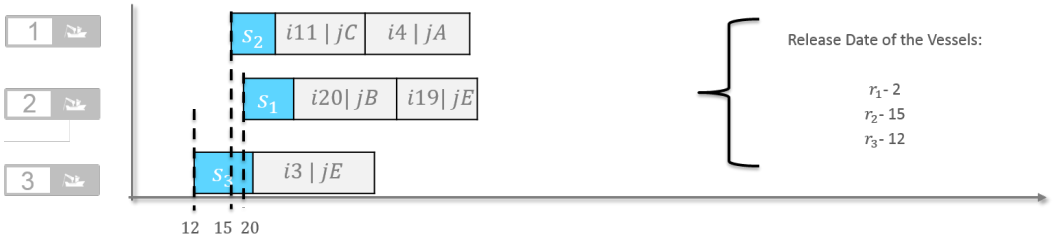

Figure 4.7: Fifth Allocation on Schedule Construction Procedure

Source: The Author

4.2.1 Constructive Heuristic Decision

To decide in which order the operations will be allocated, Abu-Marrul et al. (2020b) define a set of dispatching rules. For each instance, we run the Constructive Heuristic with all dispatching rules and select the five with the minimum weighted completion time. One important characteristic of our

problem is that our instances only consider weights of jobs, but in the Weighted Shortest Processing Time (WSPT) and Weighted Minimum Completion Time (WMCT) dispatching rules the weights of operations are considered. In the WSPT rule, the operations are ranked by their weight divided by their processing times, while the WMCT rule ranks the operations by their minimum possible completion time divided by their weights. To deal with that that the authors propose five different techniques to estimate the weights of operations:

- `MAX`: Maximum weight among associated jobs, computed as $w_i = \max_{j \in \mathcal{N}_i} w_j$
- `SUM`: Sum of the weights among associated jobs, computed as $w_i = \sum_{j \in \mathcal{N}_i} w_j$
- `AVG`: Average weight among associated jobs, computed as $w_i = \sum_{j \in \mathcal{N}_i} w_j / |\mathcal{N}_i|$
- `WAVG`: Weighted average weight among associated jobs, computed as $w_i = \sum_{j \in \mathcal{N}_i} w_j / |\mathcal{O}_j|$
- `WAVGA`: `WAVG` adjusted at each iteration by the set of non-scheduled operations.

The Weighted Adaptive Average (`WAVGA`) strategy divides the weight of the jobs for each associated operation. When one of these operations is scheduled, its estimated weight is equally distributed by all unscheduled operations of that job, raising their weight. That strategy manipulates the method to select the unscheduled operations associated to a job when most of its associated operations are already scheduled.

The authors tested six dispatching rules as candidates to rank the operations in the Constructive Heuristic. The priority value π_i is the comparison criteria to decide the next operation to schedule. At each iteration, the operation i with the largest π_i is selected. T_i is defined as the minimum possible completion time among the eligible machines for each operation i, and is computed as $T_i = \min_{k \in \mathcal{M}_i} C_k$. The other parameters included in the dispatching rules are defined in Chapter 3. As the original dispatching rules do not consider family setup times or batches, the authors adapted the rules including these parameters. The following dispatching rules are considered:

- `ERD`: Earliest Release Date $\pi_i = 1/r_i$.
- `SPT`: Shortest Processing Time $\pi_i = 1/p_i$.
- `LPT`: Longest Processing Time $\pi_i = p_i$.
- `MCT`: Minimum Completion Time $\pi_i = \max(T_i, r_i) + p_i + s_{fi}$.

– `WSPT`: Weighted Shortest Processing Time $\pi_i = w_i/p_i$.
– `WMCT`: Weighted Minimum Completion Time $\pi_i = [\max(T_i, r_i) + p_i + s_{fi}]/w_i$.

We run the Schedule Construction Procedure with all dispatching rules and all weight estimators for each dispatching rule that consider operations weights. After that, we store the five best different solutions. This solutions will serve as input for the multi-start strategy in our further Iterated Local Search.

4.3 Neighborhoods

After finding an initial solution for the problem via the Schedule Construction Procedure we need to define the Local Search neighborhood structures. These procedures define how the algorithm will perform movements in the current solution in order to improve it. The next subsections describe the four neighborhoods that will guide our further Local Search. Before explaining them, we will introduce the erase and insert functions that are the foundation of the neighborhoods movements, since all of them consist on inserting and/or erasing elements in the schedule. Both movements take constant time plus one evaluation of the schedule, then we can conclude that they run in linear time.

4.3.1 Erase Function

The erase function defines the complete set of rules that must be respected when erasing a position in the schedule. The function depends of the element contained in the position that we want to erase. We use the index p to define the position in the vector which is equivalent to a position in a machine schedule (operation or setup). Here we will use the notation of operations family O_f, without indicating the job associated to the operation because in the erase/insertion analysis it is not relevant and would pollute the illustrations. We just present the family of each operation and setup time because that is the main information to define how the insertion and removal of elements will proceed. There are two possible cases where the erase function applies: if p is an operation in a single-operation batch or if p is an operation in a multiple-operations batch. If p is a setup time, the element is not erased.

Furthermore, if position p is an operation in a single-operation batch, the function erases the operation in position p and also the setup in position $p-1$.

It occurs because it would generate a batch containing just a setup time, which is not allowed in our problem. An illustration of this case is exposed in Figure 4.8.

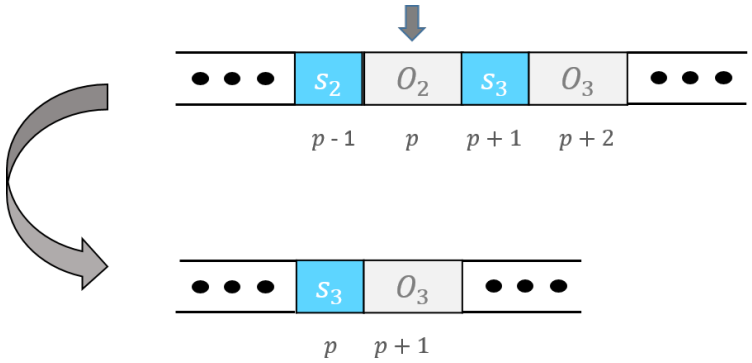

Figure 4.8: Erase Function - Single Operation Batch

Source: The Author

If p is an operation in a multiple-operations batch the erase function just delete this operation. It occurs because the removal of the operation does not result in an invalid batch composition (when a batch does not contain a setup preceding an operation). This behavior is explained in Figure 4.9.

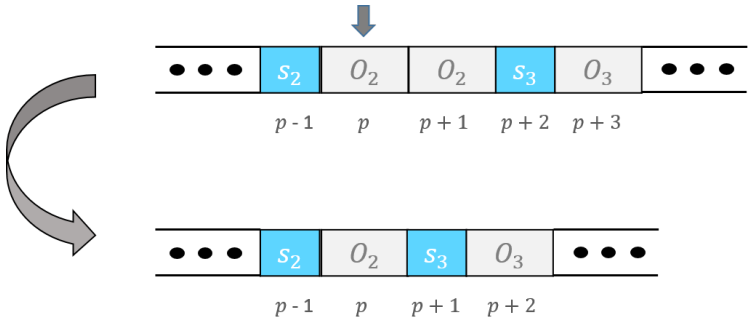

Figure 4.9: Erase Function - Multiple Operation Batch

Source: The Author

4.3.2 Insert Function

The insert function defines the complete set of rules to insert one operation in our schedule. Depending on the operation that will be inserted and its target position, the function has different outcomes. The procedure consists on inserting a predefined operation, from an specific family, in the schedule. There are five possible cases depending on the position where the operation

will be inserted. In the illustrative examples we assume that we are inserting an operation from family 3 in a different position in the schedule, denoted O_3. We use the index p to define the position in the vector which is equivalent to a position in a machine schedule (operation or setup).

– The first case is when the position p is the first position of the schedule in the machine. In this case, we insert the setup time of the same family as the operation that will be inserted in position p and the operation in position $p+1$. An illustrative example is shown in Figure 4.10.

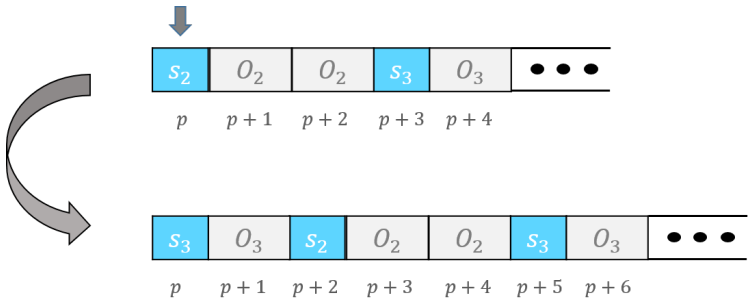

Figure 4.10: Insert Function - case 1

Source: The Author

– The second case is when position $p-1$ is a setup of the same family of the operation to be inserted. In this case, we insert the operation in position p. An illustrative example is shown in Figure 4.11.

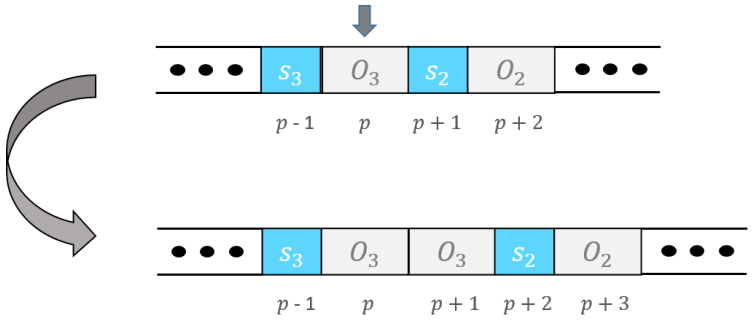

Figure 4.11: Insert Function - case 2

Source: The Author

– The third case is when position $p-1$ is an operation of a family different from the operation that will be inserted, and position p is a setup. In this case, we insert the setup of the same family as the operation to be inserted in position p and the operation in position $p+1$. An illustrative example is shown in Figure 4.12.

– The forth case is when position $p-1$ is an operation of a family different from the operation that will be inserted and position p is also an operation.

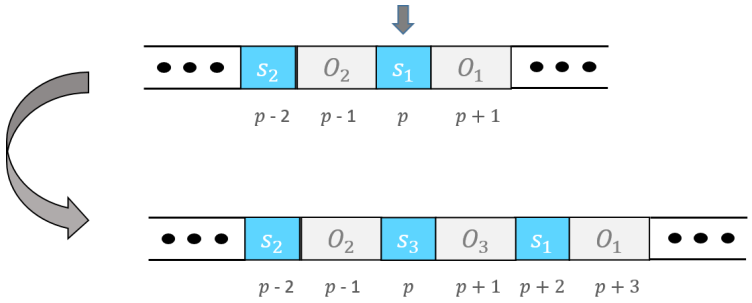

Figure 4.12: Insert Function - case 3

Source: The Author

In this case, we insert the setup of the same family as the operation to be inserted in position p, the operation in position $p+1$ and the setup of the family from the operation in $p+2$. An illustrative example is shown in Figure 4.13.

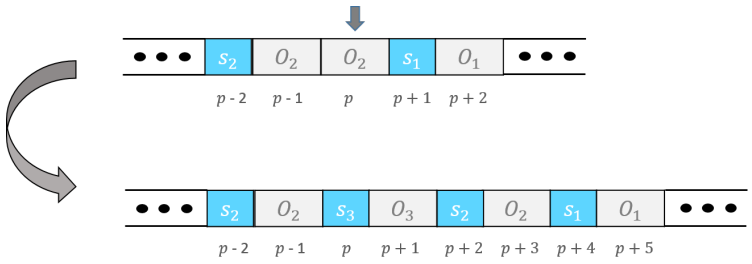

Figure 4.13: Insert Function - case 4

Source: The Author

– The fifth case is when position $p-1$ is a setup from a family different from the operation that will be inserted. In this case we insert the setup of the same family as the operation that will be inserted in position $p-1$ and the operation in position p. An illustrative example is shown in Figure 4.14.

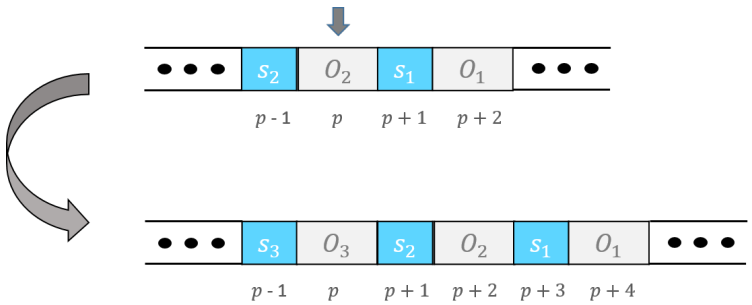

Figure 4.14: Insert Function - case 5

Source: The Author

4.3.3 Relocate Neighborhood

Once initial solutions are obtained by the Schedule Construction Procedure, it is possible to perform movements within these schedules to reach new solutions. The first neighborhood that we explore is the Relocate. The procedure consists on swiping the machine's positions, selecting each operation (one per iteration), and trying to insert it in a different position (in the same machine or in a different one). Depending on which position the operation is moved to, we have to insert one or more setup times in different positions to respect the family constraints.

To explain this method, we describe two different steps of the algorithm: the removal of setups/operations and the insertion of setups/operations using the erase and insert functions explained in sections 4.3.1 and 4.3.2.

The Relocate neighborhood arguments are: the machine and position of the operation to be relocated in the current schedule (m_1, p_1) and the machine and position where this operation will be inserted (m_2, p_2). The procedure consists in using the erase function in position p_1 of machine m_1, returning the number of elements erased. Then it inserts the operation in machine m_2 using the insert function and evaluates: if machine m_1 is different from machine m_2 or position p_1 is higher than position p_2, the operation is inserted in position p_2. Otherwise, the operation is inserted in position p_2 minus the number of elements erased by the erase function. This happens because as we erase elements in our schedule, the indexes change, moving the position p backwards. An illustrative example of each case is exposed in Figures 4.15 and 4.16.

4.3.4 Swap Neighborhood

The other neighborhood proposed in this work is the Swap. It consists on swiping the machines, selecting two operation per iteration, and changing their positions. Depending on the families of the selected operations more than one element (setup times or operations) need to be erased or inserted.

The Swap neighborhood arguments are: the machine and position of the operation to be swapped (m_1, p_1), which we will call operation 1, and the machine and position of the other operation to be swapped (m_2, p_2), which we will call operation 2. The procedure starts by erasing these operations using the erase function. Then it inserts operation 2 in the machine m_1 at position p_1 minus (the number of elements erased when erasing operation 1)

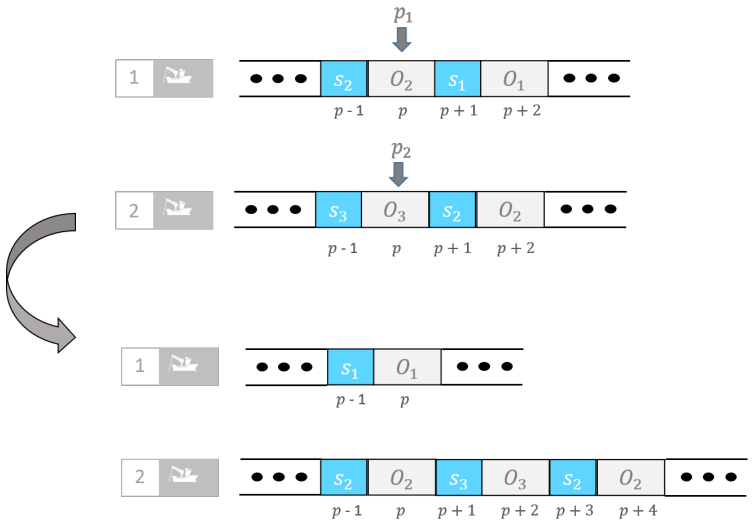

Figure 4.15: Relocate Neighborhood - case 1

Source: The Author

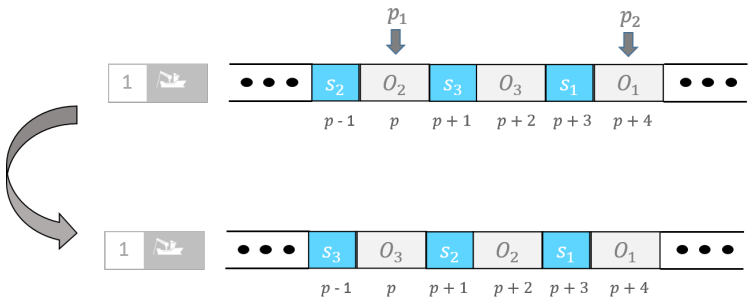

Figure 4.16: Relocate Neighborhood - case 2

Source: The Author

minus 1. Operation 1 is inserted in machine 2, but there are two possible positions for it to be inserted: if machine m_1 is different than machine m_2, this operation is inserted in position p_2 minus (the number of elements erased when erasing operation 2) plus 1. If machine m_1 is the same as machine m_2, this operation is inserted in position p_2 minus (the number of elements erased when erasing operation 2) plus 1 plus (the number of elements inserted when inserting operation 2 in machine m_1) - (the number of elements erased when erasing operation 1). Again, these particular cases are explained because when erasing or inserting elements, we change the indexes, and the position p can go backwards or forward. An illustrative example of each case is exposed in Figures 4.17 and 4.18.

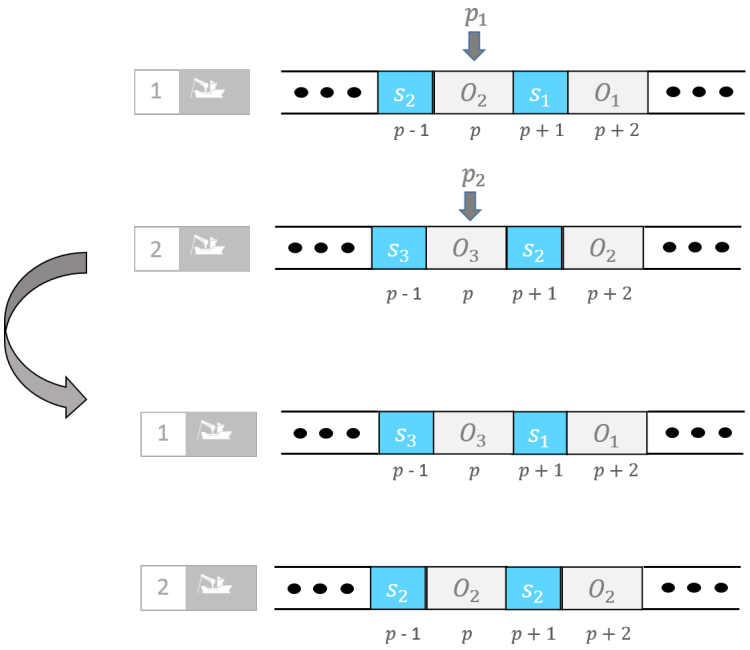

Figure 4.17: Swap Neighborhood - case 1

Source: The Author

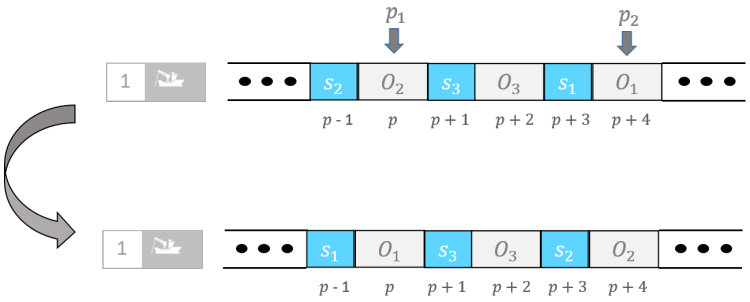

Figure 4.18: Swap Neighborhood - case 2

Source: The Author

4.3.5 Setup Insertion and Setup Removal Neighborhoods

Besides the two neighborhoods explained in the previous sections, we defined new strategies to fully understand the effect of inserting and removing just setups in our current schedule.

The first of these neighborhoods is the Setup Insertion. It works by swiping the machines searching for two consecutive operations of the same family. Once a pair of subsequent operations of the same family is found, we insert a setup of the corresponding family between them. An illustrative example is show in Figure 4.19.

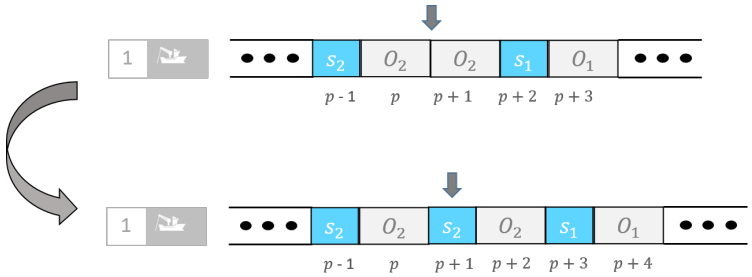

Figure 4.19: Setup Insertion

Source: The Author

The second neighborhood is the Setup Removal. It works by searching in the machines a sequence of: operation - setup time - operation, all from the same family. Then it removes this intermediate setup, transforming two batches in a single one. An illustrative example is exposed in Figure 4.20.

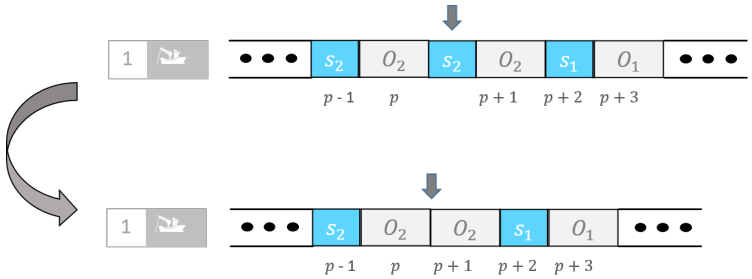

Figure 4.20: Setup Removal

Source: The Author

These neighborhoods are especially important when combined to the infeasibility strategy that will be explained in the next subsection. The Setup Removal creates batches of multiple operations, that may disrespect the occupation constraints, helping to reach infeasible solution space. The Setup Insertion is crucial to return to the feasible solution space, by dividing long batches that disrespect occupation constraints into feasible batches. It is important to notice that these neighborhoods must be combined with the Swap and Relocate neighborhoods, in order to avoid returning to the same initial solution, since they have the exact opposite effect in the current schedule.

4.3.6 Infeasibility Strategy

One important consideration on the proposed neighborhood structures is that we relax some constraints and keep some constraints hard. Basically, we relax the occupation constraints, allowing batches to have operations that exceed

the machine occupation limit. The reason to allow this violation on constraints is to enlarge the solution space, going to infeasible space and returning to feasible solutions that we could not reach if we did not pass by infeasible solutions before. The family and eligibility constraints are kept hard because the movements to reach feasible solutions would only go back to previously evaluated solutions. The infeasible solutions are penalized by the formula:

$$p(s) = \alpha c(s)$$

The penalized solution is denoted by $p(s)$, the current solution, $c(s)$, and α is the penalty factor. The initial value of α is 100 and was obtained by testing values from 10 to 200, with steps of size 10. The main idea is to increase the penalty for infeasible solutions that are reached, avoiding the search to go far from the feasible space. For that, α is enlarged in 10% at each step where an infeasible solution is considered and reduced in 5% when a feasible solution is reached. That way we can manipulate α to be in a reasonable value and not diverge from the feasible solutions space.

4.4 Local Search

After defining the neighborhood structures and infeasibility strategy we propose a local search that is used to recurrently find improved solutions for the problem. There are two main characteristics of this Local Search, which are described in the subsections that follow: the First Improvement rule and the Random Variable Neighborhood Descent (RVND).

4.4.1 First Improvement

The proposed Local Search follows the First Improvement rule. This means that the algorithm stops when it finds the first improved solution when searching in a specific neighborhood. The other possible strategy is to implement a Best Improvement rule, which evaluates all possible neighbors solutions and then selects the best of them. We chose the First Improvement rule in order to save time and computational effort while performing the Local Search.

4.4.2 Random Variable Neighborhood Descent

The other important characteristic of our local search is the Random Variable Neighborhood Descent (RVND). While performing the local search we list all the neighborhood structures detailed in the previous subsections and randomly select one of them. If the solution provided is better than the current solution, we update the current solution and shuffle again the list with neighborhood structures, including all four neighborhoods. If the solution provided is worse than the current solution we remove the used neighborhood structure from the list and randomly select the next. This algorithm goes on until all neighborhood structures have been removed from the list or when a improved solution is achieved. In Algorithm 2 the pseudocode for the RVND is described.

Algorithm 2 Random Variable Neighborhood Descent

```
1: sol ← const_sol
2: while Stop Criteria is not met do
3:     Neighborhoods ← {Swap, Relocate, SetupInsertion, SetupRemoval}
4:     while Neighborhoods not empty do
5:         Randomly select an element n from Neighborhoods
6:         Neighborhoods ← Neighborhoods − {n}
7:         sol′ ← LocalSearch(sol, n)
8:         if sol′.value < sol.value then
9:             sol ← sol′
10:            break
11:        end if
12:    end while
13: end while
14: return sol
```


4.5 Iterated Local Search

After we described the neighborhood structures and the local search characteristics, we are able to introduce the proposed Iterated Local Search for the identical parallel machine scheduling problem. This metaheuristic consists on first running the Schedule Construction Procedure method explained in subsection 4.2.1 and storing the five best different solutions, in decreasing order. These solutions will be used as input in a multi-start strategy. In each multi-start loop, we define the Schedule Construction Procedure solution and set it as our initial solution. Then the method starts the Iterated Local Search procedure by setting it as our best and current solution, and run the local

search. After it, we use a random function to select two machines and positions to perturb the current solution. Then we run the Local Search procedures again and evaluate: if a better solution is found and it is feasible, it is stored as the best solution and the current solution is updated; if we find a better solution but it is infeasible then only the current solution is updated; if the reached solution is worse than our best solution we use a simulated annealing acceptance criteria that will be explained in section 4.5.2 to define if we update our current solution or not. Another characteristic of the Iterated Local Search is the starting point strategy. If we have λ iterations without improving our best solution, we set the current solution as our best solution, avoiding solutions to diverge from good solutions that already have been reached.

After that, the next round of the multi-start procedure is performed, by setting the next best solution from the Constructive Decision as the current solution and repeating the whole Iterated Local Search algorithm. This multi-start is done with the five best solutions reached in the Constructive Decision, and the best solution among all solutions achieved in the whole method is stored as the final result. The pseudocode of the Iterated Local Search is presented in Algorithm 3.

Algorithm 3 Iterated Local Search Procedure

```
1: best_sol ← ∞
2: for i in # of Restarts do
3:     no_improve ← 0
4:     sol ← const_sol(i)
5:     cur_sol ← LocalSearch(sol)
6:     while T ≥ T_f do
7:         no_improve ← no_improve + 1
8:         sol ← perturb(cur_sol)
9:         sol' ← LocalSearch(sol)
10:        if sol'.value < cur_sol.value || AcceptanceCriteria(sol') then
11:            cur_sol ← sol'
12:        end if
13:        if sol'.value < best_sol.value && FeasibleSolution(sol') then
14:            best_sol ← sol'
15:            no_improve ← 0
16:        end if
17:        if no_improve = λ then
18:            cur_sol ← best_sol
19:            no_improve ← 0
20:        end if
21:    end while
22: end for
23: return best_sol
```

4.5.1 Perturbation

We defined the number of elements that we will perturb in our current solution, in each iteration, and how this perturbation will occur. To perturb the solutions we randomly choose two positions and two machines and one of the neighborhoods: Swap or Relocate. If the Swap structure is chosen, we change these operations position. If the Relocate structure is chosen, we relocate the operation from the first machine and position to the second.

4.5.2 Acceptance Criteria

In this work we used the simulated annealing acceptance criteria, proposed by Kirkpatrick et al. (1983) to decide whether or not we accept worse solutions in our iterated local search. This criteria is based on defining a probability to decide if we accept a worse solution at each iteration. This probability is defined by the formula:

$$p(T) = \exp(\frac{-dif}{T})$$

dif is defined as $sol - cur\_sol$ so worse solutions that are closer to the current solution have better chances to be accepted. T is the current temperature, that is updated at each iteration, being multiplied by the decreasing factor δ. The initial temperature T_i usually starts with a large value, in order to diversify the solution acceptance in the beginning of the method and at each iteration it is decreased to have less chance of accepting poorer solutions. Based on Pisinger and Røpke (2007), we defined the initial temperature T_i and final temperature T_f. The initial temperature T_i is set to accept with 0.5 probability solutions that are 40% worse than the current solution provided by the constructive heuristic following the formula in equation 4-1.

$$T_i = -\frac{0.4 \times cur\_sol}{\ln(0.5)} \tag{4-1}$$

We use the final temperature T_f to define the end of the iterated local search, so we stop it when T is the temperature that accept with 0.5 probability solutions that are 0.01% worse than the current solution, following the formula in equation 4-2.

$$T_f = -\frac{0.0001 \times cur\_sol}{\ln(0.5)} \tag{4-2}$$

As we want our iterated local search to run η iterations, we define the decreasing factor δ to decrease the initial temperature T_i to reach the stop temperature value T_f in η iterations, following the formula in equation 4-3.

$$\delta = \left(\frac{T_f}{T_i}\right)^{(\frac{1}{\eta})} \tag{4-3}$$

5
Computational Experiments

In this section, we describe all computational experiments performed and provide analysis. We tested our Iterated Local Search metaheuristic with the whole method described in Section 4.5 and defined it as the `ILS - Complete` (ils). We also ran tests for seven variations, changing the neighborhoods by enabling just the swap neighborhood, which we call `ILS - OnlySwap` (ils-os); enabling just the relocate neighborhood, which we call `ILS - OnlyRelocate` (ils-or); disabling only the swap neighborhood, the `ILS - NoSwap` (ils-ns); disabling only the relocate neighborhood, the `ILS - NoRelocate` (ils-nr); disabling only the setup insertion neighborhood, the `ILS - NoSetupInsertion` (ils-nsi); disabling only the setup removal neighborhood, the `ILS - NoSetupRemoval` (ils-nsr); and the whole ILS method but disabling the infeasibility strategy, the `ILS - NoInfeasibility` (ils-ni).

Since we have random methods on our perturbation procedures, simulated annealing acceptance criteria and variable neighborhood descent, we defined a set of ten different seeds that are used as inputs in our tests, running ten times each method in each instance. We ran the experiments on a computer with 64 GB of RAM and Intel Core i7-8700K CPU of 3.70GHz, using C++ for developing the metaheuristic and running Linux.

5.1
Instances Description

All tests were performed on a set of 72 PLSV instances, proposed by Abu-Marrul et al. (2020a), and available in Abu-Marrul et al. (2019), with $|\mathcal{M}| = \{4, 8\}$, and $|\mathcal{O}| = \{15, 25, 50\}$. Each combination of the number of Machines and Operations defines a group. Group 1 is defined as $|\mathcal{O}| = \{15\}$, $|\mathcal{M}| = \{4\}$. Group 2 as $|\mathcal{O}| = \{15\}$, $|\mathcal{M}| = \{8\}$. Group 3 as $|\mathcal{O}| = \{25\}$, $|\mathcal{M}| = \{4\}$. Group 4 as $|\mathcal{O}| = \{25\}$, $|\mathcal{M}| = \{8\}$. Group 5 as $|\mathcal{O}| = \{50\}$, $|\mathcal{M}| = \{4\}$. Group 6 as $|\mathcal{O}| = \{50\}$, $|\mathcal{M}| = \{8\}$.

Each group contains 12 instances, formed by the combination of 3 input factors in the instance generation: the release date factor $\alpha = \{0.25, 0.50, 0.75\}$, which for lower values sets the release dates closer to the beginning of the

time horizon; the eligibility factor $\beta = \{0.7, 0.9\}$, which for values closer to one generate more number of eligible operations to a machine; and the association factor $\gamma = \{0.05, 0.15\}$, defining the probability of associating a specific job to an operation.

5.2 Parameters

The Iterated Local Search algorithm has three important parameters that are tuned by testing a set of values to these parameters. The first of them is the number of iterations η. We tested the method running from 100 to 1000 iterations, with steps of size 100 and decided to stop the algorithm after 500 iterations, since the results rarely improved after it. Another parameter tuned is the number of iterations without improvement, λ, to define the starting point strategy. We tested λ from 5% to 50% of the number of iterations, with steps of size 5% and the best outcomes for the method were achieved with λ = 10%. The third parameter is the number of elements that we perturb in each iteration. It is defined as 15% of the number of operations of the current instance, rounding up this value if it is not an integer. This value was tuned after a sensitivity analysis, testing all possible percentages going from 5% to 50% with steps of size 5%. Perturbing a small number of operations did not diversify the solution space and a large number of operations set the current solution too far from the best solution achieved.

5.3 Results

In this section, we present the final results of our tests. As mentioned earlier we compare our complete method with variations that disable some neighborhoods and the infeasibility strategy. The main criteria that we compare the methods are the computational time and relative percentage deviation, computed as $RPD = \left(\frac{TWC-BKS}{BKS}\right) \times 100$. Where TWC is the total weighted completion time obtained by the method in the specific instance and the BKS is the best known solution. The BKS for our instances was obtained by the mathematical formulations proposed by Abu-Marrul et al. (2020a), with computational time of six hours.

Since we run ten times each instance for each method, we gathered the minimum (RPD^-), maximum (RPD^+) and average ($\overline{RPD}$) RPD. We present the RPD standard deviation (σ_{RPD}), indicating the methods that have more stable behaviour. The RPD^-_{LB} computed as $RPD^-_{LB} = \left(\frac{LB-BKS}{BKS}\right) \times 100$, where

LB is the Lower Bound of the instance, indicating how far from the Lower Bound the method is in an specific instance. We also defined the number of improvements ($\#Imp$) for each method, that consists on, for each method and group, counting the number of solutions reached that are equal or better than the BKS. As each group has twelve instances that we ran ten times each, the maximum number of improvements for each method in each group is 120. The $\%Imp$ is obtained by dividing $\#Imp$ by 120. Another relevant consideration that is not described in the results tables is that the four methods, `ILS - Complete`, `ILS - NoInfeasibility`, `ILS - NoSetupInsertion` and `ILS - NoSetupRemoval` were able to find or improve the previous BKS, at least once, in all 72 instances. The complete results for each instance is presented in this thesis appendix A.

Table 5.1: Results for Group 1

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\#Imp$	$\%Imp$	$\overline{Time}$
$\|\mathcal{O}\| = \{15\}$ $\|\mathcal{M}\| = \{4\}$	ils	**0.000**	0.005	**0.121**	0.024	**0**	115	95.8%	0.658
	ils-ni	**0.000**	0.003	**0.121**	0.019	**0**	117	97.5%	0.667
	ils-nr	**0.000**	0.363	5.658	0.786	**0**	55	45.8%	0.270
	ils-ns	**0.000**	3.602	6.236	2.198	**0**	10	8.3%	0.349
	ils-nsi	**0.000**	0.003	**0.121**	0.019	**0**	117	97.5%	0.686
	ils-nsr	**0.000**	**0.001**	**0.121**	**0.011**	**0**	**119**	**99.2%**	0.690
	ils-or	**0.000**	3.570	6.236	2.175	**0**	10	8.3%	0.352
	ils-os	**0.000**	0.926	5.960	1.364	**0**	40	33.3%	**0.226**

In Table 5.1, we see how the different methods performance in the first group of instances. Since Group 1 is formed by small instances, the BKS in that group are the proved optimal solutions. We can see that the `ILS - Complete`, `ILS - NoInfeasibility`, `ILS - NoSetupInsertion` and `ILS - NoSetupRemoval` have good performances in this group of instances, reaching the optimal solution in more than 95% of the tests and have very similar computational time. The `ILS - NoSetupRemoval` has the best performance among them all, reaching the optimal solution in 99% of the tests and having the best average RPD and RPD standard deviation.

Table 5.2: Results for Group 2

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\#Imp$	$\%Imp$	$\overline{Time}$
$\|\mathcal{O}\| = \{15\}$ $\|\mathcal{M}\| = \{8\}$	ils	**0.000**	0.117	3.180	0.463	**0**	108	90.0%	0.741
	ils-ni	**0.000**	**0.061**	**1.196**	**0.197**	**0**	105	87.5%	0.745
	ils-nr	**0.000**	0.928	5.783	1.175	**0**	22	18.3%	0.298
	ils-ns	**0.000**	2.561	8.234	2.423	**0**	10	8.3%	0.387
	ils-nsi	**0.000**	0.090	2.174	0.302	**0**	103	85.8%	0.746
	ils-nsr	**0.000**	0.065	2.174	0.267	**0**	**110**	**91.7%**	0.747
	ils-or	**0.000**	2.854	8.234	2.417	**0**	10	8.3%	0.381
	ils-os	**0.000**	1.038	4.722	1.185	**0**	24	20.0%	**0.252**

In Table 5.2 we present the results related to the second group of instances, which the BKS solutions are again the proved optimal solutions. In this group,

the same four methods have good performance, reaching the optimal solutions in more than 85% of the tests. In this group, it is not clear which is the best method. Again the `ILS - NoSetupRemoval` reaches more optimal solutions than the other methods, but if we look the RPD average and, especially, the standard deviation indicating that the method is more stable and reaches less often poorer solutions, the `ILS - NoInfeasibility` is the best method, reaching, in average, better solutions and also in the worst case scenario.

Table 5.3: Results for Group 3

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\#Imp$	$\%Imp$	$\overline{Time}$
$\|\mathcal{O}\| = \{25\}$ $\|\mathcal{M}\| = \{4\}$	ils	**-2.084**	-0.509	**0.165**	0.664	**0**	**109**	**90.8%**	2.902
	ils-ni	**-2.084**	-0.508	**0.165**	**0.658**	**0**	**109**	**90.8%**	2.839
	ils-nr	**-2.084**	0.240	2.883	0.920	0.704	33	27.5%	1.028
	ils-ns	0.110	3.210	5.277	1.511	1.085	0	0.0%	1.119
	ils-nsi	**-2.084**	-0.507	0.269	0.660	**0**	108	90.0%	3.041
	ils-nsr	**-2.084**	**-0.523**	0.269	0.667	**0**	107	89.2%	2.945
	ils-or	0.110	2.845	5.159	1.530	1.085	0	0.0%	1.163
	ils-os	-1.679	0.405	2.555	0.918	0.704	27	22.5%	**0.970**

In Table 5.3 we present the results among the third group of instances. From this group forward, since the instances are larger, the BKS target solutions are not optimal, so we are able to find better solutions than the previous BKS and consequently negative RPDs. The computational times for that group are slightly higher. The same four methods have the best performances, reaching BKS or better solutions in about 90% of the tests and are almost tied in all RPD metrics, with a very small advantage to the `ILS - NoInfeasibility` in the standard deviation and to the `ILS - NoSetupRemoval` in the average. Regarding the worst case scenario the `ILS - Complete` and `ILS - NoInfeasibility` are the best.

Table 5.4: Results for Group 4

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\#Imp$	$\%Imp$	$\overline{Time}$
$\|\mathcal{O}\| = \{25\}$ $\|\mathcal{M}\| = \{8\}$	ils	**-2.362**	-0.473	0.946	0.693	**0.002**	104	86.7%	3.381
	ils-ni	**-2.362**	-0.409	**0.678**	0.668	**0.002**	98	81.7%	3.275
	ils-nr	-1.938	0.671	3.077	1.112	0.064	23	19.2%	1.153
	ils-ns	2.151	4.197	6.520	1.390	3.314	0	0.0%	1.300
	ils-nsi	**-2.362**	**-0.477**	0.783	0.709	**0.002**	100	83.3%	3.490
	ils-nsr	**-2.362**	-0.457	0.802	**0.663**	**0.002**	**109**	**90.8%**	3.401
	ils-or	2.054	4.523	6.811	1.302	3.611	0	0.0%	1.300
	ils-os	-1.938	0.884	3.505	1.107	0.064	16	13.3%	**1.072**

In Table 5.4, the forth group of instances is exposed. Again the four methods have good performances, reaching or beating the target method in more than 80% of the tests. In terms of RPD average, the `ILS - NoSetupInsertion` has the best performance, almost tied with the `ILS - Complete`. In terms of standard deviation, the `ILS - NoInfeasibility` and the `ILS - NoSetupRemoval`

are the best, having a more predictable behavior. In terms of worst case scenario, the `ILS - NoInfeasibility` leads.

Table 5.5: Results for Group 5

Instance	**Method**	RPD^{-}	$\overline{RPD}$	RPD^{+}	σ_{RPD}	RPD^{-}_{LB}	$\#Imp$	$\%Imp$	$\overline{Time}$
$\|\mathcal{O}\| = \{50\}$ $\|\mathcal{M}\| = \{4\}$	ils	-7.918	-3.539	-0.650	1.948	3.553	**120**	**100.0%**	38.929
	ils-ni	-7.703	-3.431	-0.661	1.883	**3.408**	**120**	**100.0%**	30.582
	ils-nr	-6.895	-2.426	0.717	**1.850**	3.960	112	93.3%	11.853
	ils-ns	-2.976	1.047	4.561	2.213	8.954	40	33.3%	**7.284**
	ils-nsi	**-7.934**	-3.500	**-0.663**	1.893	3.575	**120**	**100.0%**	40.645
	ils-nsr	-7.910	**-3.544**	-0.602	1.940	3.508	**120**	**100.0%**	40.808
	ils-or	-2.777	1.039	4.303	1.992	8.751	30	25.0%	7.719
	ils-os	-7.004	-2.262	0.711	1.857	4.422	107	89.2%	12.920

In Table 5.5, the fifth group of instances is presented. In groups 5 and 6, we have 50 operations per instance, leading to considerably larger computational times. In this group, besides the four methods with the presumably good performances, the `ILS - NoRelocate` and `ILS - OnlySwap` also find equal or better solutions than the BKS in about 90% of the tests. In larger instances, the Iterated Local Search recurrently finds improved solutions, and the four leading methods of the other groups find the previous BKS solutions or better in all tests. In average, the `ILS - Complete` has the better results but the variance is larger, while the `ILS - NoInfeasibility` has a more stable behavior. In terms of best case scenario, the `ILS - NoSetupInsertion` has the best performance and in the worst case scenario the `ILS - NoSetupRemoval` is the best.

Table 5.6: Results for Group 6

Instance	**Method**	RPD^{-}	$\overline{RPD}$	RPD^{+}	σ_{RPD}	RPD^{-}_{LB}	$\#Imp$	$\%Imp$	$\overline{Time}$
$\|\mathcal{O}\| = \{50\}$ $\|\mathcal{M}\| = \{8\}$	ils	-11.021	**-5.194**	**-1.586**	2.638	**3.081**	**120**	**100.0%**	40.286
	ils-ni	-11.112	-5.039	-1.424	2.638	3.438	**120**	**100.0%**	31.035
	ils-nr	-10.287	-3.865	1.947	2.952	4.134	110	91.7%	11.702
	ils-ns	-5.829	-0.904	2.348	2.342	8.538	80	66.7%	**7.586**
	ils-nsi	-10.809	-5.162	-1.365	2.659	3.693	**120**	**100.0%**	41.913
	ils-nsr	**-11.429**	-5.174	-1.554	2.691	3.318	**120**	**100.0%**	41.677
	ils-or	-5.101	-0.936	2.048	**2.302**	7.155	80	66.7%	7.922
	ils-os	-9.857	-3.656	2.044	2.919	3.979	110	91.7%	12.361

In Table 5.6, we observe that the results of Group 6 are similar to the previous on Group 5. The computational time are slightly higher since we have more machines. In terms of average, the `ILS - Complete` has the best performance and in terms of standard deviation, it is almost tied with the `ILS - NoInfeasibilty`. In terms of best case scenario, the `ILS - NoSetupRemoval` reached the best solution and in terms of worst case scenario, the `ILS - NoSetupInsertion` has the best result.

Another important way to analyse the described methods and solutions achieved is by using boxplots of the RPD for each method. The graphic is

exposed in Figure 5.1, where all methods are described and the database contains all groups of instances. Analysing the graphic, we can see that the `ILS - NoSwap` and the `ILS - OnlyRelocate` are the worst methods, finding positive RPDs in average. This analysis indicates how important the swap neighborhood is to the metaheuristic, where the methods that do not enable this neighborhood have notably the worst performances.

The `ILS - NoRelocate` and the `ILS - OnlySwap` are considerably better than the methods that do not consider the swap neighborhood, but still have positive or null RPD averages. Indicating that not only the swap neighborhood is crucial to the metaheuristic but also the combination of swap and relocate neighborhoods.

As we saw in the first results tables, the methods `ILS - Complete`, `ILS - NoInfeasibility`, `ILS - NoSetupInsertion` and `ILS - NoSetupRemoval` have similar performances. In terms of RPD median the `ILS - NoSetupRemoval` has a slight advantage among them all, while the `ILS - Complete` has the lower fence. In terms of worst case scenario the `ILS - NoInfeasibility` leads as the best method.

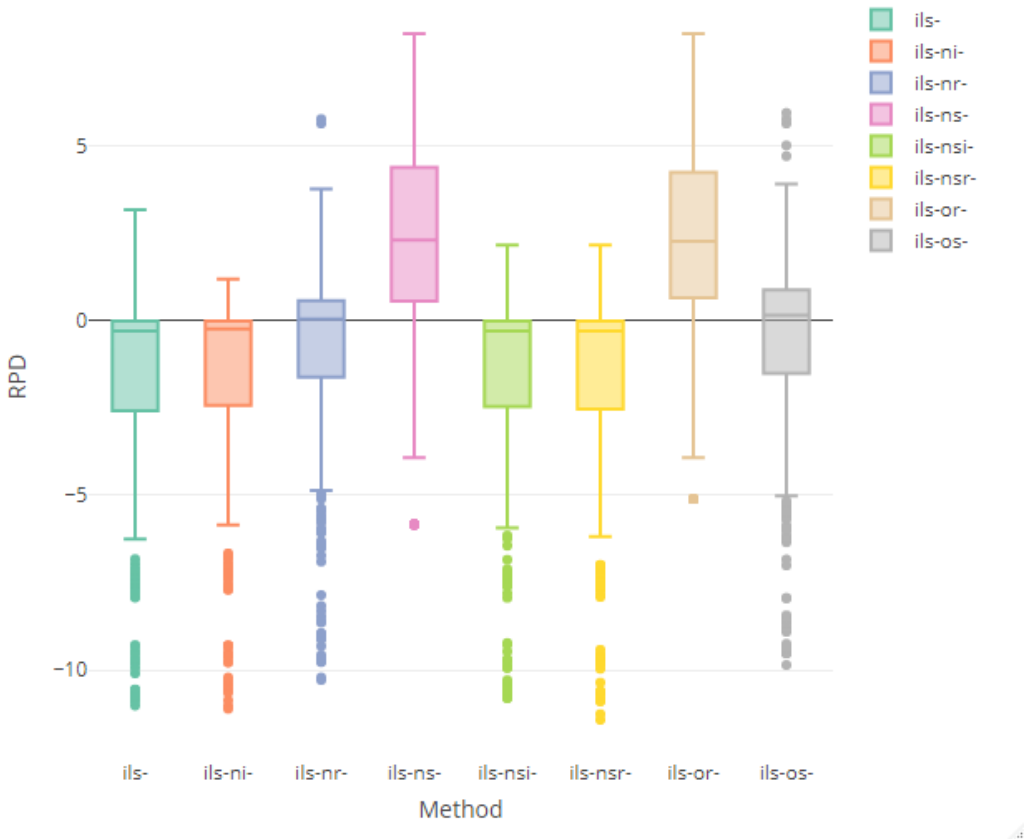

Figure 5.1: Boxplot - All Groups

Another way to compare our methods is to compare the average RPD and average computational time. If one method is faster than the other and also have lower average RPD, we can affirm that this method is dominant. In Figure 5.2, we can see that the method `ILS - NoSwap` is dominant when compared to the `ILS - OnlyRelocate`, the `ILS - NoRelocate` is dominant regarding the `ILS - OnlySwap` and the `ILS - NoSetupRemoval` dominates the `ILS - NoSetupInsertion`. With that analysis it is notable that the dominated methods can be discarded, since there are other faster methods that reach better outcomes.

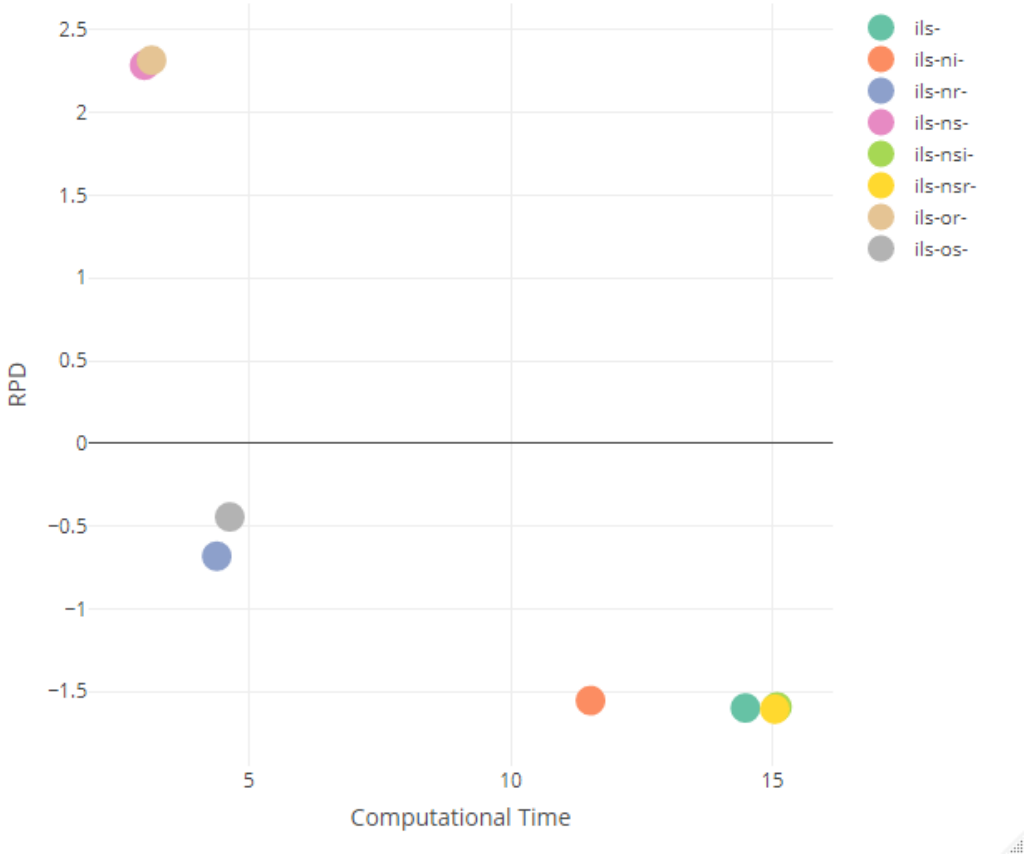

Figure 5.2: Dominance Among Methods

5.4 Statistical Tests

After analysing the results and graphics, we notice an advantage for four of the proposed methods, they are: `ILS - Complete`, `ILS - NoInfeasibility`, `ILS - NoSetupInsertion` and `ILS - NoSetupRemoval`. To define if we have significant statistical differences between the methods, we ran the ANOVA statistical test, using the software RStudio, within all pairs of methods. We

defined a significance level or 5%, so to reject the null hypothesis that the two methods do not have statistical significant differences the p-value obtained must be equal or lower than 0.05. The results of the statistical tests are shown in Table 5.7.

Table 5.7: ANOVA Tests Within All Pairs of Methods

Pair of Methods	p**_value**	**Statistical Difference**
ils-nr/ils	0.000	Yes
ils-ns/ils	0.000	Yes
ils-or/ils	0.000	Yes
ils-os/ils	0.000	Yes
ils-nr/ils-ni	0.000	Yes
ils-ns/ils-ni	0.000	Yes
ils-or/ils-ni	0.000	Yes
ils-os/ils-ni	0.000	Yes
ils-ns/ils-nr	0.000	Yes
ils-nsi/ils-nr	0.000	Yes
ils-nsr/ils-nr	0.000	Yes
ils-or/ils-nr	0.000	Yes
ils-nsi/ils-ns	0.000	Yes
ils-nsr/ils-ns	0.000	Yes
ils-os/ils-ns	0.000	Yes
ils-or/ils-nsi	0.000	Yes
ils-os/ils-nsi	0.000	Yes
ils-or/ils-nsr	0.000	Yes
ils-os/ils-nsr	0.000	Yes
ils-os/ils-or	0.000	Yes
ils-ni/ils	1.000	No
ils-nsi/ils	1.000	No
ils-nsr/ils	1.000	No
ils-nsi/ils-ni	1.000	No
ils-nsr/ils-ni	1.000	No
ils-os/ils-nr	0.624	No
ils-or/ils-ns	1.000	No
ils-nsr/ils-nsi	1.000	No

As we can expect based on the previous results and graphics, there are significant statistical difference between all pairs formed by one of the four worst methods (`ILS - NoSwap`, `ILS - OnlySwap`, `ILS - NoRelocate` and `ILS - OnlyRelocate`) and one of the four best methods (`ILS - Complete`, `ILS - NoInfeasibility`, `ILS - NoSetupInsertion` and `ILS - NoSetupRemoval`), which we could observe in the boxplots graphics, where the outcomes of the four best methods are considerably better than the four worst methods results.

Among the pairs formed by two of the four best methods, there are no

significant statistical difference between them when analysing all groups of instances. It is possible that in larger, more complex instances the differences between these methods become more notable.

There is also no significant difference between the pair of methods `ILS - OnlySwap`/`ILS - NoRelocate` and `ILS - OnlyRelocate`/`ILS - NoSwap`. It is reasonable since these pairs of methods have the same main neighborhood structures of our problem: the Swap or Relocate, but not both.

Among the four worst methods we can see that the ones that consider the Swap neighborhood are notably better than the ones that consider the Relocate neighborhood. We can observe that there are statistical differences between these pairs of methods (`ILS - NoSwap`/`ILS - NoRelocate`, `ILS - OnlyRelocate`/ `ILS - NoRelocate`, `ILS - OnlySwap`/ `ILS - NoSwap` and `ILS - OnlySwap`/`ILS - OnlyRelocate`).

6
Conclusions and Future Work

The objective of this work of proposing a metaheuristic to solve identical parallel machine scheduling problems was achieved. The presented ILS have found all the current Best Known Solutions in very competitive computational times for all presented instances. The sensitivity analyses obtained by testing the methods enabling and disabling neighborhoods and infeasibility strategies also showed which ones contribute more to the whole method and in which groups of instances they are more or less relevant.

As the results show, the infeasibility strategy does not seem to be very effective in the smaller instances, in the first four groups. Also, these strategies usually are more effective when the search goes on for a longer time. One recommended future work for this problem is to run the methods in more complex and larger instances, where probably the infeasible spaces will be better utilized. Generating these more complex instances would be a good contribution, since the PLSV instanes are limited to 50 operations so far.

Other neighborhoods could be implemented in our current Local Search and even other local search based metaheuristics can be simply coded from the presented work and to create a framework of metaheuristics for this problem is the next goal. Also it is possible to improve the neighborhood's efficiency by using pre-processing techniques, which could be an interesting future work.

The secondary objective of this thesis was also fulfilled as the instances were based on real-data from a studied company. The method was able to reach high quality solutions for those instances, indicating that it can maximize the company oil production. There are some limitations for this work, since this real-data-based instances have simulated attributes, so one possible future work is to test the method in the real-data and analyse the results to guide decision-making for the company.

As the problem described in this study is very particular, we limited our tests to the mentioned instances, but one possible future work is to test the method in other identical parallel machine scheduling problems, with similar but not equal problems characteristics. Probably some adaptations in the code would be necessary to adapt to the different characteristics, but it is a possible

extension for this work.

About the problem, there are some attributes that were simplified or excluded from this thesis because they were not in the scope of our objectives. In this problem, we are not considering due dates for the operations, jobs and machines, but in the real-life problem application those are important factors. Also, we were aiming only to minimize weighted completion time, but the company is also interested in other objectives such as minimizing tardiness, minimizing fleet and others. The extension of this work to achieve these objectives would be a relevant possible future work.

At last, all of this thesis parameters were considered as deterministic. We know that in real-life applications, most of these parameters include uncertainty, specially in the operations processing times that depend on climate conditions, the operations release dates that depend on equipment arrival and environmental licenses, and the machines release dates depending on vessels contract features. Knowing that, one very relevant future work is proposing a stochastic version of this problem, including simulation methods to better estimate the mentioned attributes.

Bibliography

Abu-Marrul, V., Martinelli, R., and Hamacher, S. (2019). Instances for the plsv scheduling problem. https://doi.org/10.17771/PUCRio.ResearchData.45799.

Abu-Marrul, V., Martinelli, R., and Hamacher, S. (2020a). Scheduling pipe laying support vessels with non-anticipatory family setup times and intersections between sets of operations. *International Journal of Production Research*, to appear.

Abu-Marrul, V., Mecler, D., Martinelli, R., Hamacher, S., and Gribkovskaia, I. (2020b). Heuristics for scheduling pipe-laying support vessels: An identical parallel machine scheduling approach. In *17th International Conference on Project Management and Scheduling*.

Azizoglu, M. and Webster, S. (2003). Scheduling parallel machines to minimize weighted flowtime with family set-up times. *International Journal of Production Research*, 41(6):1199–1215.

Bremenkamp, L. H. (2017). Metodologia baseada em programação matemática para alocação e sequenciamento em embarcações plsv. Master's thesis, Departamento de Engenharia Industrial - Pontifícia Universidade Católica do Rio de Janeiro. in Portuguese.

Chen, Z.-L. and Powell, W. B. (2003). Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Research Logistics*, 50(7):823–840.

Ciavotta, M., Meloni, C., and Pranzo, M. (2016). Speeding up a rollout algorithm for complex parallel machine scheduling. *International Journal of Production Research*, 54:4993–5009.

Dipak, L. and Gupta, J. N. D. (2018). An improved cuckoo search algorithm for scheduling jobs on identical parallel machines. *Computers & Industrial Engineering*, 126:348–360.

Dunstall, S. and Wirth, A. (2005a). A comparison of branch-and-bound algorithms for a family scheduling problem with identical parallel machines. *European Journal of Operational Research*, 167(2):283–296.

Dunstall, S. and Wirth, A. (2005b). Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research*, 32(9):2479–2491.

Kaplan, S. and Rabadi, G. (2011). Simulated annealing and metaheuristic for randomized priority search algorithms for the aerial refuelling parallel machine

scheduling problem with due date-to-deadline windows and release times. *Engineering Optimization*, 45:67–87.

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

Lee, C. (2017). A dispatching rule and a random iterated greedy metaheuristic for identical parallel machine scheduling to minimize total tardiness. *International Journal of Production Research*, 56:1–17.

Liao, C.-J., Chao, C.-W., and Chen, L.-C. (2012). An improved heuristic for parallel machine weighted flowtime scheduling with family set-up times. *Computers & Mathematics with Applications*, 63(1):110–117.

Mehdizadeh, E., Tavakkoli-Moghaddam, R., and Yazdani, M. (2015). A vibration damping optimization algorithm for a parallel machines scheduling problem with sequence-independent family setup times. *Applied Mathematical Modelling*, 39(22):6845–6859.

Mendes, A. S.and Mueller, F. M., França, P. M., and Moscato, P. (2002). Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning and Control*, 13:143–154.

Min, L. and Cheng, W. (1999). A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13:399–402.

Mokotoff, E. (2004). An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research*, 152:758–769.

Nguyen, N., Yalaoui, F., Amodeo, L., C. H., and P., T. (2018). Total completion time minimization for machine scheduling problem under time windows constraints with jobs' linear processing rate function. *Computers and Operations Research*, 90:110–124.

Omar, M. K. and Teo, S. C. (2006). Minimizing the sum of earliness/tardiness in identical parallel machines schedule with incompatible job families: An improved MIP approach. *Applied Mathematics and Computation*, 181(2):1008–1017.

Pacciarelli, D., Meloni, C., and Pranzo, M. (2011). Models and methods for production scheduling in the pharmaceutical industry. *International Series in Operations Research & Management Science*, 152:429–459.

Pisinger, D. and Røpke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European journal of operational research*, 120(2):228–249.

Ruiz, R. and Maroto, C. (2005). An improved water flow-like algorithm for order acceptance and scheduling with identical parallel machines. *European Journal of Operational Research*, 165:479–494.

Schaller, J. E. (2014). Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering*, 72:274–281.

Shin, H. J. and Leon, V. J. (2004). International Journal of Production Research Scheduling with product family set-up times: an application in TFT LCD manufacturing Scheduling with product family set-up times: an application in TFT LCD manufacturing. *International Journal of Production Research*, 42(20):4235–4248.

Unlu, Y. and Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800.

Van Der Zee, D.-J. (2015). Family-based dispatching with parallel machines. *International Journal of Production Research*, 53:5837–5856.

Webster, S. and Azizoglu, M. (2001). Dynamic programming algorithms for scheduling parallel machines with family setup times. *Computers & Operations Research*, 28(2):127–137.

Wu, G., Cheng, C., Yang, H., and Chena, C. (2018). A comprehensive review and evaluation of permutation flowshop heuristics. *Applied Soft Computing*, 71:1072–1084.

A
Detailed Results of the Methods for each Instance

In this appendix, we present the complete results of our tests, that were summarized in Chapter 5. Table A.1 presents, for each instance and method, the main metrics that we defined to evaluate the results.

Since we run each method ten times, with ten different seeds, we gathered the minimum (RPD^-), maximum (RPD^+) and average ($\overline{RPD}$) RPD. RPD is computed as $RPD = \left(\frac{TWC-BKS}{BKS}\right) \times 100$. Where TWC is the total weighted completion time obtained by the method in the specific instance and the BKS is the best known solution. We present the RPD standard deviation (σ_{RPD}) and the RPD^-_{LB} computed as $RPD^-_{LB} = \left(\frac{LB-BKS}{BKS}\right) \times 100$, where LB is the Lower Bound of the instance, indicating how far from the Lower Bound the method is in an specific instance. At last the $\overline{Time}$ is presented indicating the average computational time to run the method in each instance.

Table A.1: Complete Results for each Method and Instance

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o15_n5_q3_m4_111	ils	0.000	0.000	0.000	0.000	0.000	0.60
o15_n5_q3_m4_111	ils-ni	0.000	0.000	0.000	0.000	0.000	0.55
o15_n5_q3_m4_111	ils-ns	0.000	0.000	0.000	0.000	0.000	0.34
o15_n5_q3_m4_111	ils-nr	0.000	0.000	0.000	0.000	0.000	0.215
o15_n5_q3_m4_111	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.586
o15_n5_q3_m4_111	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.527
o15_n5_q3_m4_111	ils-os	0.000	0.000	0.000	0.000	0.000	0.173
o15_n5_q3_m4_111	ils-or	0.000	0.000	0.000	0.000	0.000	0.362
o15_n5_q3_m4_112	ils	0.000	0.000	0.000	0.000	0.000	0.83
o15_n5_q3_m4_112	ils-ni	0.000	0.000	0.000	0.000	0.000	0.80
o15_n5_q3_m4_112	ils-ns	0.583	0.583	0.583	0.000	0.583	0.33
o15_n5_q3_m4_112	ils-nr	0.000	0.238	0.596	23.754	0.000	0.281
o15_n5_q3_m4_112	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.752
o15_n5_q3_m4_112	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.767
o15_n5_q3_m4_112	ils-os	0.000	0.555	1.076	31.975	0.000	0.239
o15_n5_q3_m4_112	ils-or	0.583	0.583	0.583	0.000	0.583	0.359
o15_n5_q3_m4_121	ils	0.000	0.000	0.000	0.000	0.000	0.62
o15_n5_q3_m4_121	ils-ni	0.000	0.000	0.000	0.000	0.000	0.58

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o15_n5_q3_m4_121	ils-ns	2.294	2.294	2.294	0.000	2.294	0.31
o15_n5_q3_m4_121	ils-nr	0.360	0.743	2.161	52.262	0.360	0.235
o15_n5_q3_m4_121	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.569
o15_n5_q3_m4_121	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.571
o15_n5_q3_m4_121	ils-os	0.360	2.082	3.241	87.708	0.360	0.199
o15_n5_q3_m4_121	ils-or	2.361	2.361	2.361	0.000	2.361	0.350
o15_n5_q3_m4_122	ils	0.000	0.000	0.000	0.000	0.000	0.68
o15_n5_q3_m4_122	ils-ni	0.000	0.000	0.000	0.000	0.000	0.70
o15_n5_q3_m4_122	ils-ns	5.797	5.797	5.797	0.000	5.797	0.31
o15_n5_q3_m4_122	ils-nr	0.836	2.104	5.658	141.002	0.836	0.244
o15_n5_q3_m4_122	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.655
o15_n5_q3_m4_122	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.645
o15_n5_q3_m4_122	ils-os	0.836	3.568	5.960	174.141	0.836	0.222
o15_n5_q3_m4_122	ils-or	5.797	5.797	5.797	0.000	5.797	0.372
o15_n5_q3_m4_131	ils	0.000	0.000	0.000	0.000	0.000	0.56
o15_n5_q3_m4_131	ils-ni	0.000	0.000	0.000	0.000	0.000	0.56
o15_n5_q3_m4_131	ils-ns	0.730	0.730	0.730	0.000	0.730	0.33
o15_n5_q3_m4_131	ils-nr	0.182	0.283	0.365	5.676	0.182	0.227
o15_n5_q3_m4_131	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.548
o15_n5_q3_m4_131	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.531
o15_n5_q3_m4_131	ils-os	0.365	0.520	1.186	35.917	0.365	0.163
o15_n5_q3_m4_131	ils-or	0.730	0.730	0.730	0.000	0.730	0.344
o15_n5_q3_m4_132	ils	0.000	0.000	0.000	0.000	0.000	0.76
o15_n5_q3_m4_132	ils-ni	0.000	0.000	0.000	0.000	0.000	0.66
o15_n5_q3_m4_132	ils-ns	5.793	5.793	5.793	0.000	5.793	0.41
o15_n5_q3_m4_132	ils-nr	0.000	0.000	0.000	0.000	0.000	0.273
o15_n5_q3_m4_132	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.654
o15_n5_q3_m4_132	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.658
o15_n5_q3_m4_132	ils-os	0.000	0.020	0.197	6.641	0.000	0.213
o15_n5_q3_m4_132	ils-or	4.689	4.689	4.689	0.000	4.689	0.343
o15_n5_q3_m4_211	ils	0.000	0.000	0.000	0.000	0.000	0.65
o15_n5_q3_m4_211	ils-ni	0.000	0.000	0.000	0.000	0.000	0.57
o15_n5_q3_m4_211	ils-ns	4.886	4.886	4.886	0.000	4.886	0.47
o15_n5_q3_m4_211	ils-nr	0.000	0.173	1.734	48.383	0.000	0.233
o15_n5_q3_m4_211	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.569
o15_n5_q3_m4_211	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.541
o15_n5_q3_m4_211	ils-os	1.032	2.489	3.117	68.950	1.032	0.178
o15_n5_q3_m4_211	ils-or	4.886	4.886	4.886	0.000	4.886	0.390
o15_n5_q3_m4_212	ils	0.000	0.000	0.000	0.000	0.000	0.92
o15_n5_q3_m4_212	ils-ni	0.000	0.000	0.000	0.000	0.000	0.81
o15_n5_q3_m4_212	ils-ns	4.651	4.651	4.651	0.000	4.651	0.31

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o15_n5_q3_m4_212	ils-nr	0.106	0.106	0.106	0.000	0.106	0.315
o15_n5_q3_m4_212	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.822
o15_n5_q3_m4_212	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.823
o15_n5_q3_m4_212	ils-os	0.106	0.240	0.912	15.915	0.106	0.267
o15_n5_q3_m4_212	ils-or	4.651	4.651	4.651	0.000	4.651	0.337
o15_n5_q3_m4_221	ils	0.000	0.000	0.000	0.000	0.000	0.60
o15_n5_q3_m4_221	ils-ni	0.000	0.000	0.000	0.000	0.000	0.70
o15_n5_q3_m4_221	ils-ns	2.193	2.193	2.193	0.000	2.193	0.34
o15_n5_q3_m4_221	ils-nr	0.000	0.000	0.000	0.000	0.000	0.237
o15_n5_q3_m4_221	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.561
o15_n5_q3_m4_221	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.635
o15_n5_q3_m4_221	ils-os	0.000	0.000	0.000	0.000	0.000	0.208
o15_n5_q3_m4_221	ils-or	2.193	2.193	2.193	0.000	2.193	0.361
o15_n5_q3_m4_222	ils	0.000	0.000	0.000	0.000	0.000	0.69
o15_n5_q3_m4_222	ils-ni	0.000	0.000	0.000	0.000	0.000	0.71
o15_n5_q3_m4_222	ils-ns	4.806	4.806	4.806	0.000	4.806	0.32
o15_n5_q3_m4_222	ils-nr	0.000	0.096	0.193	14.757	0.000	0.239
o15_n5_q3_m4_222	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.592
o15_n5_q3_m4_222	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.646
o15_n5_q3_m4_222	ils-os	0.000	0.077	0.193	14.459	0.000	0.220
o15_n5_q3_m4_222	ils-or	4.806	4.806	4.806	0.000	4.806	0.341
o15_n5_q3_m4_231	ils	0.000	0.060	0.121	6.852	0.000	0.68
o15_n5_q3_m4_231	ils-ni	0.000	0.036	0.121	6.280	0.000	0.78
o15_n5_q3_m4_231	ils-ns	5.253	5.253	5.253	0.000	5.263	0.32
o15_n5_q3_m4_231	ils-nr	0.195	0.195	0.195	0.000	0.205	0.249
o15_n5_q3_m4_231	ils-nsi	0.000	0.036	0.121	6.280	0.009	0.585
o15_n5_q3_m4_231	ils-nsr	0.000	0.012	0.121	4.111	0.009	0.644
o15_n5_q3_m4_231	ils-os	0.195	0.322	1.460	43.007	0.205	0.212
o15_n5_q3_m4_231	ils-or	5.904	5.904	5.904	10754.000	5.914	0.357
o15_n5_q3_m4_232	ils	0.000	0.000	0.000	0.000	0.000	0.56
o15_n5_q3_m4_232	ils-ni	0.000	0.000	0.000	0.000	0.000	0.60
o15_n5_q3_m4_232	ils-ns	6.236	6.236	6.236	0.000	6.236	0.33
o15_n5_q3_m4_232	ils-nr	0.000	0.417	1.550	97.437	0.000	0.195
o15_n5_q3_m4_232	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.585
o15_n5_q3_m4_232	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.543
o15_n5_q3_m4_232	ils-os	0.000	1.243	2.737	162.076	0.000	0.175
o15_n5_q3_m4_232	ils-or	6.236	6.236	6.236	0.000	6.236	0.310
o15_n5_q3_m8_111	ils	0.000	0.000	0.000	0.000	0.000	0.79
o15_n5_q3_m8_111	ils-ni	0.000	0.000	0.000	0.000	0.000	0.71
o15_n5_q3_m8_111	ils-ns	2.369	2.369	2.369	0.000	2.369	0.49
o15_n5_q3_m8_111	ils-nr	0.813	0.813	0.813	0.000	0.813	0.263

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o15_n5_q3_m8_111	ils-nsi	0.000	0.007	0.070	0.949	0.000	0.681
o15_n5_q3_m8_111	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.698
o15_n5_q3_m8_111	ils-os	0.813	0.876	1.393	7.846	0.813	0.235
o15_n5_q3_m8_111	ils-or	2.508	2.508	2.508	0.000	2.508	0.392
o15_n5_q3_m8_112	ils	0.000	0.000	0.000	0.000	0.000	0.87
o15_n5_q3_m8_112	ils-ni	0.000	0.000	0.000	0.000	0.000	0.78
o15_n5_q3_m8_112	ils-ns	8.234	8.234	8.234	0.000	8.234	0.39
o15_n5_q3_m8_112	ils-nr	0.568	0.592	0.807	4.111	0.568	0.297
o15_n5_q3_m8_112	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.712
o15_n5_q3_m8_112	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.768
o15_n5_q3_m8_112	ils-os	1.064	1.097	1.284	3.706	1.064	0.246
o15_n5_q3_m8_112	ils-or	8.234	8.234	8.234	0.000	8.234	0.403
o15_n5_q3_m8_121	ils	0.000	0.000	0.000	0.000	0.000	0.65
o15_n5_q3_m8_121	ils-ni	0.000	0.000	0.000	0.000	0.000	0.63
o15_n5_q3_m8_121	ils-ns	0.000	0.000	0.000	0.000	0.000	0.46
o15_n5_q3_m8_121	ils-nr	0.000	0.000	0.000	0.000	0.000	0.258
o15_n5_q3_m8_121	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.625
o15_n5_q3_m8_121	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.537
o15_n5_q3_m8_121	ils-os	0.000	0.000	0.000	0.000	0.000	0.197
o15_n5_q3_m8_121	ils-or	0.000	0.000	0.000	0.000	0.000	0.379
o15_n5_q3_m8_122	ils	0.000	0.000	0.000	0.000	0.000	0.68
o15_n5_q3_m8_122	ils-ni	0.000	0.000	0.000	0.000	0.000	0.79
o15_n5_q3_m8_122	ils-ns	1.649	1.649	1.649	0.000	1.649	0.35
o15_n5_q3_m8_122	ils-nr	0.000	0.000	0.000	0.000	0.000	0.244
o15_n5_q3_m8_122	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.653
o15_n5_q3_m8_122	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.640
o15_n5_q3_m8_122	ils-os	0.000	0.000	0.000	0.000	0.000	0.233
o15_n5_q3_m8_122	ils-or	1.649	1.649	1.649	0.000	1.649	0.356
o15_n5_q3_m8_131	ils	0.000	0.318	3.180	43.639	0.000	0.71
o15_n5_q3_m8_131	ils-ni	0.000	0.028	0.138	2.530	0.000	0.69
o15_n5_q3_m8_131	ils-ns	3.042	3.042	3.042	0.000	3.042	0.35
o15_n5_q3_m8_131	ils-nr	1.659	2.461	3.180	20.357	1.659	0.255
o15_n5_q3_m8_131	ils-nsi	0.000	0.014	0.138	1.897	0.000	0.653
o15_n5_q3_m8_131	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.615
o15_n5_q3_m8_131	ils-os	2.627	2.627	2.627	0.000	2.627	0.205
o15_n5_q3_m8_131	ils-or	3.042	3.042	3.042	0.000	3.042	0.398
o15_n5_q3_m8_132	ils	0.000	0.015	0.149	3.479	0.000	0.73
o15_n5_q3_m8_132	ils-ni	0.000	0.000	0.000	0.000	0.000	0.87
o15_n5_q3_m8_132	ils-ns	1.465	1.465	1.465	0.000	1.465	0.41
o15_n5_q3_m8_132	ils-nr	0.109	0.396	0.760	20.060	0.109	0.245
o15_n5_q3_m8_132	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.667

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o15_n5_q3_m8_132	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.695
o15_n5_q3_m8_132	ils-os	0.109	0.355	0.760	21.994	0.109	0.216
o15_n5_q3_m8_132	ils-or	4.002	4.002	4.002	0.000	4.002	0.374
o15_n5_q3_m8_211	ils	0.000	0.000	0.000	0.000	0.000	0.79
o15_n5_q3_m8_211	ils-ni	0.000	0.000	0.000	0.000	0.000	0.83
o15_n5_q3_m8_211	ils-ns	1.631	1.631	1.631	0.000	1.631	0.35
o15_n5_q3_m8_211	ils-nr	0.533	0.561	0.564	0.316	0.533	0.293
o15_n5_q3_m8_211	ils-nsi	0.000	0.022	0.220	2.214	0.000	0.685
o15_n5_q3_m8_211	ils-nsr	0.000	0.022	0.220	2.214	0.000	0.735
o15_n5_q3_m8_211	ils-os	0.533	0.668	1.317	7.761	0.533	0.229
o15_n5_q3_m8_211	ils-or	1.380	1.380	1.380	0.000	1.380	0.372
o15_n5_q3_m8_212	ils	0.000	0.000	0.000	0.000	0.000	0.82
o15_n5_q3_m8_212	ils-ni	0.000	0.021	0.164	2.234	0.000	0.70
o15_n5_q3_m8_212	ils-ns	2.021	2.021	2.021	0.000	2.021	0.34
o15_n5_q3_m8_212	ils-nr	0.164	0.183	0.258	1.687	0.164	0.285
o15_n5_q3_m8_212	ils-nsi	0.000	0.049	0.164	3.381	0.000	0.701
o15_n5_q3_m8_212	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.705
o15_n5_q3_m8_212	ils-os	0.164	0.174	0.258	1.265	0.164	0.230
o15_n5_q3_m8_212	ils-or	2.021	2.021	2.021	0.000	2.021	0.359
o15_n5_q3_m8_221	ils	0.000	1.058	2.174	44.109	0.000	0.87
o15_n5_q3_m8_221	ils-ni	0.000	0.645	1.196	15.855	0.000	0.78
o15_n5_q3_m8_221	ils-ns	1.703	1.703	1.703	0.000	1.703	0.41
o15_n5_q3_m8_221	ils-nr	0.580	1.410	2.537	33.072	0.580	0.291
o15_n5_q3_m8_221	ils-nsi	0.652	0.975	2.174	27.088	0.652	0.755
o15_n5_q3_m8_221	ils-nsr	0.000	0.743	2.174	33.879	0.000	0.670
o15_n5_q3_m8_221	ils-os	1.268	2.044	2.645	31.073	1.268	0.240
o15_n5_q3_m8_221	ils-or	2.790	2.790	2.790	0.000	2.790	0.412
o15_n5_q3_m8_222	ils	0.000	0.000	0.000	0.000	0.000	0.78
o15_n5_q3_m8_222	ils-ni	0.000	0.000	0.000	0.000	0.000	0.79
o15_n5_q3_m8_222	ils-ns	7.055	7.055	7.055	0.000	7.055	0.37
o15_n5_q3_m8_222	ils-nr	3.785	3.985	5.783	66.092	3.785	0.271
o15_n5_q3_m8_222	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.607
o15_n5_q3_m8_222	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.635
o15_n5_q3_m8_222	ils-os	3.785	3.879	4.722	30.990	3.785	0.232
o15_n5_q3_m8_222	ils-or	7.055	7.055	7.055	0.000	7.055	0.411
o15_n5_q3_m8_231	ils	0.000	0.019	0.187	4.743	0.000	0.76
o15_n5_q3_m8_231	ils-ni	0.000	0.037	0.187	6.325	0.000	0.78
o15_n5_q3_m8_231	ils-ns	0.337	0.337	0.337	0.000	0.337	0.32
o15_n5_q3_m8_231	ils-nr	0.187	0.247	0.337	5.138	0.187	0.250
o15_n5_q3_m8_231	ils-nsi	0.000	0.019	0.187	4.743	0.000	0.651
o15_n5_q3_m8_231	ils-nsr	0.000	0.000	0.000	0.000	0.000	0.690

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o15_n5_q3_m8_231	ils-os	0.187	0.304	0.412	6.075	0.187	0.218
o15_n5_q3_m8_231	ils-or	0.337	0.337	0.337	0.000	0.337	0.346
o15_n5_q3_m8_232	ils	0.000	0.000	0.000	0.000	0.000	0.86
o15_n5_q3_m8_232	ils-ni	0.000	0.000	0.000	0.000	0.000	0.81
o15_n5_q3_m8_232	ils-ns	1.229	1.229	1.229	0.000	1.229	0.41
o15_n5_q3_m8_232	ils-nr	0.000	0.492	0.702	13.506	0.000	0.305
o15_n5_q3_m8_232	ils-nsi	0.000	0.000	0.000	0.000	0.000	0.753
o15_n5_q3_m8_232	ils-nsr	0.000	0.012	0.117	1.897	0.000	0.755
o15_n5_q3_m8_232	ils-os	0.000	0.427	0.819	19.081	0.000	0.265
o15_n5_q3_m8_232	ils-or	1.229	1.229	1.229	0.000	1.229	0.369
o25_n8_q3_m4_111	ils	0.000	0.000	0.000	0.000	2.566	2.85
o25_n8_q3_m4_111	ils-ni	0.000	0.000	0.000	0.000	2.566	2.78
o25_n8_q3_m4_111	ils-ns	2.874	2.874	2.874	0.000	5.513	1.07
o25_n8_q3_m4_111	ils-nr	0.579	0.766	1.038	12.758	3.160	0.845
o25_n8_q3_m4_111	ils-nsi	0.000	0.000	0.000	0.000	2.566	2.432
o25_n8_q3_m4_111	ils-nsr	0.000	0.000	0.000	0.000	2.566	2.354
o25_n8_q3_m4_111	ils-os	0.710	0.989	1.519	19.884	3.294	0.793
o25_n8_q3_m4_111	ils-or	2.907	2.907	2.907	0.000	5.547	1.103
o25_n8_q3_m4_112	ils	-0.522	-0.344	-0.021	44.413	11.407	2.77
o25_n8_q3_m4_112	ils-ni	-0.522	-0.402	0.149	45.012	11.407	2.78
o25_n8_q3_m4_112	ils-ns	2.184	2.184	2.184	0.000	14.437	1.05
o25_n8_q3_m4_112	ils-nr	0.202	0.398	0.996	51.998	12.219	1.055
o25_n8_q3_m4_112	ils-nsi	-0.522	-0.454	-0.144	27.006	11.407	2.627
o25_n8_q3_m4_112	ils-nsr	-0.522	-0.492	-0.224	17.709	11.407	2.608
o25_n8_q3_m4_112	ils-os	0.000	0.503	1.321	88.699	11.992	0.987
o25_n8_q3_m4_112	ils-or	1.912	1.912	1.912	0.000	14.133	1.104
o25_n8_q3_m4_121	ils	-1.163	-1.098	-0.514	47.434	5.485	3.48
o25_n8_q3_m4_121	ils-ni	-1.163	-1.043	-0.514	58.836	5.485	3.39
o25_n8_q3_m4_121	ils-ns	4.997	4.997	4.997	0.000	12.059	1.16
o25_n8_q3_m4_121	ils-nr	-0.942	-0.077	0.437	97.497	5.720	1.097
o25_n8_q3_m4_121	ils-nsi	-1.163	-1.012	-0.380	73.639	5.485	3.369
o25_n8_q3_m4_121	ils-nsr	-1.163	-1.163	-1.163	0.000	5.485	3.265
o25_n8_q3_m4_121	ils-os	-0.359	-0.020	0.367	49.054	6.343	1.045
o25_n8_q3_m4_121	ils-or	0.921	0.921	0.921	0.000	7.708	1.202
o25_n8_q3_m4_122	ils	0.000	0.000	0.000	0.000	3.640	3.03
o25_n8_q3_m4_122	ils-ni	0.000	0.000	0.000	0.000	3.640	2.67
o25_n8_q3_m4_122	ils-ns	2.803	2.803	2.803	0.000	6.545	1.07
o25_n8_q3_m4_122	ils-nr	0.081	0.371	1.804	67.899	3.723	0.870
o25_n8_q3_m4_122	ils-nsi	0.000	0.000	0.000	0.000	3.640	2.953
o25_n8_q3_m4_122	ils-nsr	0.000	0.000	0.000	0.000	3.640	2.634
o25_n8_q3_m4_122	ils-os	0.081	0.239	0.814	27.475	3.723	0.779

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o25_n8_q3_m4_122	ils-or	2.642	2.642	2.642	0.000	6.378	1.158
o25_n8_q3_m4_131	ils	0.061	0.149	0.165	11.474	4.402	3.19
o25_n8_q3_m4_131	ils-ni	0.165	0.165	0.165	0.000	4.510	3.21
o25_n8_q3_m4_131	ils-ns	3.171	3.171	3.171	0.000	7.647	1.22
o25_n8_q3_m4_131	ils-nr	0.317	0.571	0.841	50.213	4.669	1.000
o25_n8_q3_m4_131	ils-nsi	0.113	0.159	0.165	5.376	4.456	3.063
o25_n8_q3_m4_131	ils-nsr	0.061	0.149	0.165	11.474	4.402	3.054
o25_n8_q3_m4_131	ils-os	0.549	0.778	1.003	50.553	4.911	0.932
o25_n8_q3_m4_131	ils-or	3.171	3.171	3.171	0.000	7.647	1.214
o25_n8_q3_m4_132	ils	0.000	0.000	0.000	0.000	0.000	2.99
o25_n8_q3_m4_132	ils-ni	0.000	0.000	0.000	0.000	0.000	2.78
o25_n8_q3_m4_132	ils-ns	1.085	1.085	1.085	0.000	1.085	1.22
o25_n8_q3_m4_132	ils-nr	0.704	1.044	1.426	69.033	0.704	0.904
o25_n8_q3_m4_132	ils-nsi	0.000	0.054	0.269	31.201	0.000	2.988
o25_n8_q3_m4_132	ils-nsr	0.000	0.081	0.269	35.745	0.000	2.669
o25_n8_q3_m4_132	ils-os	0.704	1.259	1.764	117.384	0.704	0.815
o25_n8_q3_m4_132	ils-or	1.085	1.085	1.085	0.000	1.085	1.348
o25_n8_q3_m4_211	ils	-1.392	-1.392	-1.392	0.000	9.972	2.58
o25_n8_q3_m4_211	ils-ni	-1.392	-1.392	-1.392	0.000	9.972	2.59
o25_n8_q3_m4_211	ils-ns	3.698	3.698	3.698	0.000	15.649	1.02
o25_n8_q3_m4_211	ils-nr	-1.392	-1.065	-0.206	141.346	9.972	0.985
o25_n8_q3_m4_211	ils-nsi	-1.392	-1.392	-1.392	0.000	9.972	2.404
o25_n8_q3_m4_211	ils-nsr	-1.392	-1.392	-1.392	0.000	9.972	2.437
o25_n8_q3_m4_211	ils-os	-1.392	-0.981	-0.299	128.902	9.972	0.951
o25_n8_q3_m4_211	ils-or	3.698	3.698	3.698	0.000	15.649	1.017
o25_n8_q3_m4_212	ils	-2.084	-2.084	-2.084	0.000	10.945	2.99
o25_n8_q3_m4_212	ils-ni	-2.084	-2.074	-2.024	4.175	10.945	3.19
o25_n8_q3_m4_212	ils-ns	5.277	5.277	5.277	0.000	19.285	1.13
o25_n8_q3_m4_212	ils-nr	-2.084	-1.622	-0.944	60.178	10.945	0.944
o25_n8_q3_m4_212	ils-nsi	-2.084	-2.078	-2.024	3.795	10.945	2.822
o25_n8_q3_m4_212	ils-nsr	-2.084	-2.084	-2.084	0.000	10.945	2.785
o25_n8_q3_m4_212	ils-os	-1.679	-1.295	-0.515	78.266	11.404	0.903
o25_n8_q3_m4_212	ils-or	4.922	4.922	4.922	0.000	18.883	1.182
o25_n8_q3_m4_221	ils	-0.557	-0.557	-0.557	0.000	4.360	2.81
o25_n8_q3_m4_221	ils-ni	-0.557	-0.557	-0.557	0.000	4.360	2.73
o25_n8_q3_m4_221	ils-ns	4.954	4.954	4.954	0.000	10.143	1.06
o25_n8_q3_m4_221	ils-nr	0.481	0.656	0.847	25.963	5.449	0.920
o25_n8_q3_m4_221	ils-nsi	-0.557	-0.557	-0.557	0.000	4.360	2.647
o25_n8_q3_m4_221	ils-nsr	-0.557	-0.557	-0.557	0.000	4.360	2.660
o25_n8_q3_m4_221	ils-os	0.461	0.918	1.218	50.342	5.428	0.910
o25_n8_q3_m4_221	ils-or	5.159	5.159	5.159	0.000	10.359	1.198

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o25_n8_q3_m4_222	ils	-0.462	-0.423	-0.326	12.186	10.799	2.51
o25_n8_q3_m4_222	ils-ni	-0.462	-0.407	-0.396	6.332	10.799	2.36
o25_n8_q3_m4_222	ils-ns	0.110	0.110	0.110	0.000	11.436	1.11
o25_n8_q3_m4_222	ils-nr	-0.326	0.085	0.480	63.699	10.950	0.759
o25_n8_q3_m4_222	ils-nsi	-0.396	-0.396	-0.396	0.000	10.873	2.315
o25_n8_q3_m4_222	ils-nsr	-0.462	-0.417	-0.396	7.843	10.799	2.215
o25_n8_q3_m4_222	ils-os	-0.044	0.261	0.579	58.581	11.264	0.691
o25_n8_q3_m4_222	ils-or	0.110	0.110	0.110	0.000	11.436	1.163
o25_n8_q3_m4_231	ils	0.000	0.000	0.000	0.000	2.225	2.98
o25_n8_q3_m4_231	ils-ni	0.000	0.000	0.000	0.000	2.225	2.58
o25_n8_q3_m4_231	ils-ns	4.199	4.199	4.199	0.000	6.518	1.01
o25_n8_q3_m4_231	ils-nr	0.491	1.666	2.883	223.599	2.727	0.830
o25_n8_q3_m4_231	ils-nsi	0.000	0.000	0.000	0.000	2.225	2.582
o25_n8_q3_m4_231	ils-nsr	0.000	0.000	0.000	0.000	2.225	2.499
o25_n8_q3_m4_231	ils-os	1.282	1.932	2.555	119.380	3.536	0.792
o25_n8_q3_m4_231	ils-or	4.199	4.199	4.199	0.000	6.518	1.066
o25_n8_q3_m4_232	ils	-0.403	-0.355	0.078	44.904	5.411	3.16
o25_n8_q3_m4_232	ils-ni	-0.403	-0.385	-0.217	17.393	5.411	3.02
o25_n8_q3_m4_232	ils-ns	3.169	3.169	3.169	0.000	9.192	1.15
o25_n8_q3_m4_232	ils-nr	-0.044	0.086	0.454	41.711	5.791	1.010
o25_n8_q3_m4_232	ils-nsi	-0.403	-0.403	-0.403	0.000	5.411	2.969
o25_n8_q3_m4_232	ils-nsr	-0.403	-0.403	-0.403	0.000	5.411	2.949
o25_n8_q3_m4_232	ils-os	0.075	0.273	1.112	90.068	5.917	0.985
o25_n8_q3_m4_232	ils-or	3.417	3.417	3.417	0.000	9.454	1.207
o25_n8_q3_m8_111	ils	-1.133	-0.842	-0.260	33.116	4.021	3.36
o25_n8_q3_m8_111	ils-ni	-0.926	-0.676	-0.398	26.104	4.239	3.30
o25_n8_q3_m8_111	ils-ns	2.457	2.457	2.457	0.000	7.799	1.26
o25_n8_q3_m8_111	ils-nr	-0.813	0.139	0.986	62.310	4.358	1.019
o25_n8_q3_m8_111	ils-nsi	-1.479	-0.928	-0.398	41.625	3.657	3.160
o25_n8_q3_m8_111	ils-nsr	-0.986	-0.824	-0.424	22.394	4.176	3.006
o25_n8_q3_m8_111	ils-os	0.407	0.716	1.099	25.024	5.641	0.887
o25_n8_q3_m8_111	ils-or	3.971	3.971	3.971	0.000	9.392	1.231
o25_n8_q3_m8_112	ils	-0.601	-0.539	-0.272	15.720	6.724	3.67
o25_n8_q3_m8_112	ils-ni	-0.601	-0.394	-0.012	28.001	6.724	3.47
o25_n8_q3_m8_112	ils-ns	6.520	6.520	6.520	0.000	14.369	1.31
o25_n8_q3_m8_112	ils-nr	2.322	2.676	3.077	36.907	9.862	1.220
o25_n8_q3_m8_112	ils-nsi	-0.570	-0.443	-0.272	20.951	6.757	3.554
o25_n8_q3_m8_112	ils-nsr	-0.570	-0.430	-0.272	20.024	6.757	3.457
o25_n8_q3_m8_112	ils-os	2.050	2.731	3.505	63.757	9.569	1.129
o25_n8_q3_m8_112	ils-or	6.811	6.811	6.811	0.000	14.682	1.388
o25_n8_q3_m8_121	ils	0.000	0.408	0.697	31.590	0.593	3.30

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o25_n8_q3_m8_121	ils-ni	0.105	0.459	0.678	21.419	0.699	3.21
o25_n8_q3_m8_121	ils-ns	4.963	4.963	4.963	0.000	5.586	1.45
o25_n8_q3_m8_121	ils-nr	0.687	1.206	1.642	33.791	1.285	0.998
o25_n8_q3_m8_121	ils-nsi	0.000	0.453	0.783	29.224	0.593	3.055
o25_n8_q3_m8_121	ils-nsr	0.000	0.440	0.802	33.838	0.593	3.010
o25_n8_q3_m8_121	ils-os	1.012	1.219	1.727	22.804	1.611	0.933
o25_n8_q3_m8_121	ils-or	4.648	4.648	4.648	0.000	5.269	1.411
o25_n8_q3_m8_122	ils	-0.385	-0.271	0.096	30.945	7.046	2.69
o25_n8_q3_m8_122	ils-ni	-0.254	-0.038	0.223	42.053	7.188	2.54
o25_n8_q3_m8_122	ils-ns	3.443	3.443	3.443	0.000	11.160	1.30
o25_n8_q3_m8_122	ils-nr	0.482	1.206	1.795	71.879	7.978	0.882
o25_n8_q3_m8_122	ils-nsi	-0.385	-0.161	0.304	49.313	7.046	2.505
o25_n8_q3_m8_122	ils-nsr	-0.385	-0.257	-0.157	16.248	7.046	2.476
o25_n8_q3_m8_122	ils-os	1.197	1.692	2.287	78.940	8.746	0.770
o25_n8_q3_m8_122	ils-or	6.074	6.074	6.074	0.000	13.987	1.270
o25_n8_q3_m8_131	ils	0.000	0.000	0.000	0.000	0.002	2.89
o25_n8_q3_m8_131	ils-ni	0.000	0.029	0.165	5.266	0.002	2.87
o25_n8_q3_m8_131	ils-ns	3.691	3.691	3.691	0.000	3.693	1.25
o25_n8_q3_m8_131	ils-nr	0.062	0.300	0.619	16.374	0.064	0.940
o25_n8_q3_m8_131	ils-nsi	0.000	0.025	0.186	5.797	0.002	2.794
o25_n8_q3_m8_131	ils-nsr	0.000	0.000	0.000	0.000	0.002	2.676
o25_n8_q3_m8_131	ils-os	0.062	0.524	0.835	28.055	0.064	0.884
o25_n8_q3_m8_131	ils-or	3.691	3.691	3.691	0.000	3.693	1.338
o25_n8_q3_m8_132	ils	-0.201	-0.098	0.234	27.097	1.687	3.26
o25_n8_q3_m8_132	ils-ni	-0.201	-0.154	0.167	20.966	1.687	3.16
o25_n8_q3_m8_132	ils-ns	5.121	5.121	5.121	0.000	7.109	1.27
o25_n8_q3_m8_132	ils-nr	0.568	1.023	1.872	80.359	2.471	1.029
o25_n8_q3_m8_132	ils-nsi	-0.201	-0.156	0.084	16.234	1.687	3.236
o25_n8_q3_m8_132	ils-nsr	-0.201	-0.178	-0.095	7.505	1.687	2.997
o25_n8_q3_m8_132	ils-os	0.435	1.026	1.543	63.877	2.335	0.984
o25_n8_q3_m8_132	ils-or	5.121	5.121	5.121	0.000	7.109	1.238
o25_n8_q3_m8_211	ils	-1.290	-1.057	-0.824	17.797	2.774	3.72
o25_n8_q3_m8_211	ils-ni	-0.968	-0.780	-0.466	13.937	3.110	3.87
o25_n8_q3_m8_211	ils-ns	6.046	6.046	6.046	0.000	10.413	1.25
o25_n8_q3_m8_211	ils-nr	-0.633	-0.179	0.167	24.345	3.459	1.109
o25_n8_q3_m8_211	ils-nsi	-1.290	-1.036	-0.824	15.195	2.774	3.735
o25_n8_q3_m8_211	ils-nsr	-1.290	-0.950	-0.824	10.947	2.774	3.613
o25_n8_q3_m8_211	ils-os	-0.203	0.103	0.609	20.738	3.906	1.040
o25_n8_q3_m8_211	ils-or	6.022	6.022	6.022	0.000	10.388	1.181
o25_n8_q3_m8_212	ils	-1.339	-0.854	-0.303	47.778	4.675	3.28
o25_n8_q3_m8_212	ils-ni	-0.962	-0.765	-0.215	39.981	5.076	3.44

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o25_n8_q3_m8_212	ils-ns	3.632	3.632	3.632	0.000	9.950	1.21
o25_n8_q3_m8_212	ils-nr	-0.503	0.139	0.481	45.672	5.562	1.123
o25_n8_q3_m8_212	ils-nsi	-1.339	-0.862	0.222	54.801	4.675	3.197
o25_n8_q3_m8_212	ils-nsr	-1.339	-0.754	-0.333	45.379	4.675	3.225
o25_n8_q3_m8_212	ils-os	-0.311	0.432	1.280	76.757	5.766	1.052
o25_n8_q3_m8_212	ils-or	3.795	3.795	3.795	0.000	10.122	1.146
o25_n8_q3_m8_221	ils	0.000	0.135	0.946	31.815	0.180	3.08
o25_n8_q3_m8_221	ils-ni	0.000	0.030	0.100	5.314	0.180	2.89
o25_n8_q3_m8_221	ils-ns	3.129	3.129	3.129	0.000	3.314	1.22
o25_n8_q3_m8_221	ils-nr	1.992	2.284	2.984	33.238	2.175	0.910
o25_n8_q3_m8_221	ils-nsi	0.000	0.050	0.100	5.798	0.180	2.816
o25_n8_q3_m8_221	ils-nsr	0.000	0.040	0.100	5.680	0.180	2.745
o25_n8_q3_m8_221	ils-os	1.856	2.552	3.248	53.910	2.039	0.818
o25_n8_q3_m8_221	ils-or	3.921	3.921	3.921	0.000	4.107	1.379
o25_n8_q3_m8_222	ils	-2.362	-2.176	-1.881	28.663	6.170	3.86
o25_n8_q3_m8_222	ils-ni	-2.362	-2.207	-1.959	26.232	6.170	3.60
o25_n8_q3_m8_222	ils-ns	5.813	5.813	5.813	0.000	15.060	1.37
o25_n8_q3_m8_222	ils-nr	-1.938	-1.491	-1.089	40.625	6.631	1.215
o25_n8_q3_m8_222	ils-nsi	-2.362	-2.227	-1.669	35.716	6.170	3.567
o25_n8_q3_m8_222	ils-nsr	-2.362	-2.171	-1.761	31.774	6.170	3.569
o25_n8_q3_m8_222	ils-os	-1.938	-1.332	-0.806	53.877	6.631	1.167
o25_n8_q3_m8_222	ils-or	4.958	4.958	4.958	0.000	14.129	1.398
o25_n8_q3_m8_231	ils	-0.227	-0.201	0.032	7.589	1.295	3.15
o25_n8_q3_m8_231	ils-ni	-0.227	-0.227	-0.227	0.000	1.295	2.96
o25_n8_q3_m8_231	ils-ns	2.151	2.151	2.151	0.000	3.709	1.25
o25_n8_q3_m8_231	ils-nr	0.303	0.500	0.681	13.704	1.833	1.014
o25_n8_q3_m8_231	ils-nsi	-0.227	-0.227	-0.227	0.000	1.295	3.003
o25_n8_q3_m8_231	ils-nsr	-0.227	-0.227	-0.227	0.000	1.295	2.991
o25_n8_q3_m8_231	ils-os	0.540	0.715	0.940	11.070	2.074	0.965
o25_n8_q3_m8_231	ils-or	2.054	2.054	2.054	0.000	3.611	1.231
o25_n8_q3_m8_232	ils	-0.366	-0.184	-0.059	15.526	2.657	3.64
o25_n8_q3_m8_232	ils-ni	-0.289	-0.181	-0.065	12.240	2.736	3.50
o25_n8_q3_m8_232	ils-ns	3.402	3.402	3.402	0.000	6.540	1.32
o25_n8_q3_m8_232	ils-nr	0.053	0.245	0.396	19.783	3.089	1.124
o25_n8_q3_m8_232	ils-nsi	-0.366	-0.208	-0.083	17.041	2.657	3.446
o25_n8_q3_m8_232	ils-nsr	-0.390	-0.177	-0.083	16.234	2.633	3.337
o25_n8_q3_m8_232	ils-os	0.159	0.226	0.425	13.425	3.199	1.066
o25_n8_q3_m8_232	ils-or	3.208	3.208	3.208	0.000	6.339	1.387
o50_n16_q3_m4_111	ils	-4.932	-4.656	-4.366	125.615	13.851	44.10
o50_n16_q3_m4_111	ils-ni	-5.024	-4.597	-4.080	171.786	13.741	35.20
o50_n16_q3_m4_111	ils-ns	4.239	4.239	4.239	0.000	24.835	6.40

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o50_n16_q3_m4_111	ils-nr	-3.877	-3.142	-2.284	389.740	15.115	11.535
o50_n16_q3_m4_111	ils-nsi	-4.834	-4.585	-4.286	128.645	13.969	40.604
o50_n16_q3_m4_111	ils-nsr	-5.012	-4.622	-4.414	106.700	13.756	42.881
o50_n16_q3_m4_111	ils-os	-3.746	-3.130	-2.346	283.493	15.272	12.612
o50_n16_q3_m4_111	ils-or	4.303	4.303	4.303	0.000	24.911	7.127
o50_n16_q3_m4_112	ils	-4.835	-4.394	-4.171	197.004	18.841	34.94
o50_n16_q3_m4_112	ils-ni	-4.382	-4.078	-3.619	233.386	19.407	26.96
o50_n16_q3_m4_112	ils-ns	0.200	0.200	0.200	0.000	25.128	7.61
o50_n16_q3_m4_112	ils-nr	-3.786	-2.720	-1.912	571.159	20.151	9.221
o50_n16_q3_m4_112	ils-nsi	-4.473	-4.193	-3.874	161.492	19.294	33.063
o50_n16_q3_m4_112	ils-nsr	-4.876	-4.398	-3.810	311.662	18.789	32.452
o50_n16_q3_m4_112	ils-os	-3.877	-2.362	-1.727	589.493	20.038	10.045
o50_n16_q3_m4_112	ils-or	1.024	1.024	1.024	0.000	26.158	8.334
o50_n16_q3_m4_121	ils	-2.727	-2.551	-2.364	78.245	6.735	37.53
o50_n16_q3_m4_121	ils-ni	-2.692	-2.368	-2.028	165.385	6.774	30.25
o50_n16_q3_m4_121	ils-ns	2.421	2.421	2.421	0.000	12.384	7.29
o50_n16_q3_m4_121	ils-nr	-1.881	-1.480	-1.153	173.145	7.664	11.050
o50_n16_q3_m4_121	ils-nsi	-2.692	-2.421	-2.218	103.310	6.774	34.830
o50_n16_q3_m4_121	ils-nsr	-2.785	-2.527	-2.188	114.464	6.671	34.658
o50_n16_q3_m4_121	ils-os	-1.706	-1.424	-0.891	161.060	7.856	12.115
o50_n16_q3_m4_121	ils-or	1.809	1.809	1.809	0.000	11.713	7.147
o50_n16_q3_m4_122	ils	-2.882	-2.575	-2.221	195.584	12.743	46.13
o50_n16_q3_m4_122	ils-ni	-2.908	-2.533	-2.199	212.856	12.713	34.56
o50_n16_q3_m4_122	ils-ns	0.549	0.549	0.549	0.000	16.726	8.04
o50_n16_q3_m4_122	ils-nr	-2.268	-1.848	-1.481	291.995	13.456	12.535
o50_n16_q3_m4_122	ils-nsi	-2.842	-2.601	-2.275	188.916	12.789	44.082
o50_n16_q3_m4_122	ils-nsr	-2.762	-2.586	-2.474	107.652	12.882	44.717
o50_n16_q3_m4_122	ils-os	-2.607	-1.836	-1.458	369.968	13.062	13.624
o50_n16_q3_m4_122	ils-or	0.591	0.591	0.591	0.000	16.775	8.219
o50_n16_q3_m4_131	ils	-1.640	-1.596	-1.414	64.291	5.490	40.48
o50_n16_q3_m4_131	ils-ni	-1.633	-1.550	-1.358	80.424	5.497	33.13
o50_n16_q3_m4_131	ils-ns	4.561	4.561	4.561	0.000	12.141	7.03
o50_n16_q3_m4_131	ils-nr	-1.300	-0.845	-0.332	293.344	5.854	11.393
o50_n16_q3_m4_131	ils-nsi	-1.811	-1.660	-1.558	66.000	5.307	37.760
o50_n16_q3_m4_131	ils-nsr	-1.740	-1.639	-1.516	61.654	5.383	38.415
o50_n16_q3_m4_131	ils-os	-0.905	-0.567	-0.166	214.979	6.278	12.872
o50_n16_q3_m4_131	ils-or	2.068	2.068	2.068	0.000	9.467	7.470
o50_n16_q3_m4_132	ils	-1.158	-0.889	-0.650	156.502	6.718	48.66
o50_n16_q3_m4_132	ils-ni	-1.176	-0.839	-0.661	163.383	6.698	40.52
o50_n16_q3_m4_132	ils-ns	1.487	1.487	1.487	0.000	9.573	7.60
o50_n16_q3_m4_132	ils-nr	-0.390	-0.164	0.133	168.434	7.547	13.114

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o50_n16_q3_m4_132	ils-nsi	-1.085	-0.913	-0.663	125.675	6.796	45.335
o50_n16_q3_m4_132	ils-nsr	-1.046	-0.828	-0.602	129.029	6.839	45.565
o50_n16_q3_m4_132	ils-os	-0.297	0.008	0.320	228.688	7.648	14.435
o50_n16_q3_m4_132	ils-or	2.553	2.553	2.553	0.000	10.725	8.468
o50_n16_q3_m4_211	ils	-6.248	-5.834	-5.454	244.691	9.214	35.26
o50_n16_q3_m4_211	ils-ni	-5.851	-5.489	-5.179	202.163	9.676	24.62
o50_n16_q3_m4_211	ils-ns	-2.243	-2.243	-2.243	0.000	13.878	6.57
o50_n16_q3_m4_211	ils-nr	-5.370	-4.454	-3.727	423.200	10.236	9.127
o50_n16_q3_m4_211	ils-nsi	-6.437	-5.793	-5.460	283.937	8.993	33.678
o50_n16_q3_m4_211	ils-nsr	-6.183	-5.849	-5.329	216.258	9.289	32.621
o50_n16_q3_m4_211	ils-os	-4.891	-4.405	-3.975	303.759	10.794	9.873
o50_n16_q3_m4_211	ils-or	-2.207	-2.207	-2.207	0.000	13.921	7.276
o50_n16_q3_m4_212	ils	-7.918	-7.668	-7.287	171.049	15.508	34.20
o50_n16_q3_m4_212	ils-ni	-7.703	-7.404	-7.146	184.134	15.779	23.91
o50_n16_q3_m4_212	ils-ns	-2.976	-2.976	-2.976	0.000	21.707	7.31
o50_n16_q3_m4_212	ils-nr	-6.895	-6.383	-5.947	267.820	16.792	9.764
o50_n16_q3_m4_212	ils-nsi	-7.934	-7.497	-7.183	216.401	15.488	32.247
o50_n16_q3_m4_212	ils-nsr	-7.910	-7.615	-7.230	174.608	15.518	32.735
o50_n16_q3_m4_212	ils-os	-7.004	-6.234	-5.679	363.633	16.655	10.777
o50_n16_q3_m4_212	ils-or	-2.777	-2.777	-2.777	0.000	21.958	7.257
o50_n16_q3_m4_221	ils	-2.899	-2.812	-2.571	112.682	9.361	41.81
o50_n16_q3_m4_221	ils-ni	-2.835	-2.676	-2.239	235.512	9.433	33.69
o50_n16_q3_m4_221	ils-ns	2.361	2.361	2.361	0.000	15.285	7.47
o50_n16_q3_m4_221	ils-nr	-2.159	-1.779	-1.501	226.160	10.194	11.085
o50_n16_q3_m4_221	ils-nsi	-2.899	-2.791	-2.658	84.367	9.361	40.155
o50_n16_q3_m4_221	ils-nsr	-2.899	-2.815	-2.461	144.565	9.361	40.003
o50_n16_q3_m4_221	ils-os	-1.859	-1.569	-1.213	233.009	10.531	11.862
o50_n16_q3_m4_221	ils-or	0.304	0.304	0.304	0.000	12.968	7.519
o50_n16_q3_m4_222	ils	-3.672	-3.344	-2.878	251.539	19.448	32.01
o50_n16_q3_m4_222	ils-ni	-4.176	-3.435	-3.145	321.616	18.824	26.33
o50_n16_q3_m4_222	ils-ns	2.181	2.181	2.181	0.000	26.706	7.91
o50_n16_q3_m4_222	ils-nr	-3.056	-2.169	-1.243	708.794	20.212	8.719
o50_n16_q3_m4_222	ils-nsi	-3.703	-3.262	-2.890	263.043	19.410	31.229
o50_n16_q3_m4_222	ils-nsr	-3.872	-3.389	-2.946	283.971	19.200	30.214
o50_n16_q3_m4_222	ils-os	-2.491	-1.849	-1.282	457.119	20.913	9.230
o50_n16_q3_m4_222	ils-or	2.121	2.121	2.121	0.000	26.632	7.544
o50_n16_q3_m4_231	ils	-5.078	-4.912	-4.616	142.526	3.553	43.74
o50_n16_q3_m4_231	ils-ni	-5.211	-4.957	-4.706	162.716	3.408	37.34
o50_n16_q3_m4_231	ils-ns	-0.127	-0.127	-0.127	0.000	8.954	7.34
o50_n16_q3_m4_231	ils-nr	-4.705	-4.178	-3.854	202.720	3.960	13.033
o50_n16_q3_m4_231	ils-nsi	-5.058	-4.944	-4.740	96.526	3.575	40.507

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o50_n16_q3_m4_231	ils-nsr	-5.119	-4.974	-4.764	95.978	3.508	41.505
o50_n16_q3_m4_231	ils-os	-4.281	-3.960	-3.650	185.122	4.422	14.263
o50_n16_q3_m4_231	ils-or	-0.313	-0.313	-0.313	0.000	8.751	8.013
o50_n16_q3_m4_232	ils	-1.468	-1.233	-0.862	263.335	13.337	31.36
o50_n16_q3_m4_232	ils-ni	-1.542	-1.243	-1.030	242.128	13.253	21.43
o50_n16_q3_m4_232	ils-ns	-0.092	-0.092	-0.092	0.000	14.921	7.24
o50_n16_q3_m4_232	ils-nr	-0.975	0.047	0.717	558.439	13.904	8.730
o50_n16_q3_m4_232	ils-nsi	-1.558	-1.345	-0.971	236.597	13.234	29.915
o50_n16_q3_m4_232	ils-nsr	-1.499	-1.288	-1.001	218.837	13.302	29.412
o50_n16_q3_m4_232	ils-os	-0.920	0.189	0.711	578.867	13.967	9.242
o50_n16_q3_m4_232	ils-or	2.989	2.989	2.989	0.000	18.464	8.272
o50_n16_q3_m8_111	ils	-3.743	-3.454	-3.092	76.359	9.192	31.53
o50_n16_q3_m8_111	ils-ni	-3.848	-3.214	-2.498	119.531	9.074	26.32
o50_n16_q3_m8_111	ils-ns	1.166	1.166	1.166	0.000	14.762	6.80
o50_n16_q3_m8_111	ils-nr	-2.179	-1.513	-0.855	152.625	10.967	8.740
o50_n16_q3_m8_111	ils-nsi	-3.615	-3.398	-3.211	52.243	9.337	30.892
o50_n16_q3_m8_111	ils-nsr	-3.796	-3.353	-2.807	116.083	9.133	31.069
o50_n16_q3_m8_111	ils-os	-2.277	-1.673	-1.035	145.861	10.855	9.085
o50_n16_q3_m8_111	ils-or	2.048	2.048	2.048	0.000	15.762	7.087
o50_n16_q3_m8_112	ils	-11.021	-10.658	-10.097	126.120	20.828	40.14
o50_n16_q3_m8_112	ils-ni	-11.112	-10.617	-10.216	153.189	20.705	31.70
o50_n16_q3_m8_112	ils-ns	-5.829	-5.829	-5.829	0.000	27.879	8.32
o50_n16_q3_m8_112	ils-nr	-10.287	-9.628	-8.974	216.999	21.826	10.348
o50_n16_q3_m8_112	ils-nsi	-10.809	-10.601	-10.402	72.167	21.116	38.755
o50_n16_q3_m8_112	ils-nsr	-11.429	-10.820	-10.362	162.580	20.275	38.522
o50_n16_q3_m8_112	ils-os	-9.857	-9.322	-8.796	179.366	22.409	10.772
o50_n16_q3_m8_112	ils-or	-5.101	-5.101	-5.101	0.000	28.868	9.051
o50_n16_q3_m8_121	ils	-5.219	-5.098	-4.912	28.837	8.051	38.18
o50_n16_q3_m8_121	ils-ni	-5.200	-4.971	-4.545	66.853	8.073	28.35
o50_n16_q3_m8_121	ils-ns	-1.850	-1.850	-1.850	0.000	11.891	7.31
o50_n16_q3_m8_121	ils-nr	-4.392	-4.036	-3.447	98.546	8.993	10.516
o50_n16_q3_m8_121	ils-nsi	-5.291	-5.097	-4.921	43.001	7.969	36.568
o50_n16_q3_m8_121	ils-nsr	-5.212	-5.033	-4.758	48.356	8.058	35.565
o50_n16_q3_m8_121	ils-os	-4.145	-3.618	-2.893	125.012	9.275	11.096
o50_n16_q3_m8_121	ils-or	-1.850	-1.850	-1.850	0.000	11.891	7.547
o50_n16_q3_m8_122	ils	-5.766	-5.026	-4.597	163.812	15.060	41.48
o50_n16_q3_m8_122	ils-ni	-5.151	-4.615	-4.336	127.789	15.811	30.73
o50_n16_q3_m8_122	ils-ns	-0.957	-0.957	-0.957	0.000	20.932	8.08
o50_n16_q3_m8_122	ils-nr	-4.350	-3.674	-3.317	171.749	16.788	9.962
o50_n16_q3_m8_122	ils-nsi	-5.307	-4.956	-4.485	120.964	15.620	39.283
o50_n16_q3_m8_122	ils-nsr	-5.407	-4.975	-4.611	114.504	15.498	38.406

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o50_n16_q3_m8_122	ils-os	-3.699	-3.306	-3.070	94.883	17.584	10.677
o50_n16_q3_m8_122	ils-or	-1.473	-1.473	-1.473	0.000	20.301	8.311
o50_n16_q3_m8_131	ils	-3.257	-3.008	-2.572	151.353	3.866	34.79
o50_n16_q3_m8_131	ils-ni	-3.207	-2.738	-2.239	171.065	3.919	29.12
o50_n16_q3_m8_131	ils-ns	1.870	1.870	1.870	0.000	9.370	7.71
o50_n16_q3_m8_131	ils-nr	-2.631	-1.983	-1.235	297.577	4.538	8.861
o50_n16_q3_m8_131	ils-nsi	-3.459	-2.930	-2.214	209.763	3.648	33.622
o50_n16_q3_m8_131	ils-nsr	-3.156	-2.823	-2.377	175.026	3.974	32.419
o50_n16_q3_m8_131	ils-os	-1.995	-1.647	-1.197	172.068	5.220	9.117
o50_n16_q3_m8_131	ils-or	1.870	1.870	1.870	0.000	9.370	7.839
o50_n16_q3_m8_132	ils	-5.318	-4.666	-4.230	218.757	15.355	39.37
o50_n16_q3_m8_132	ils-ni	-4.829	-4.452	-4.054	163.768	15.950	29.51
o50_n16_q3_m8_132	ils-ns	-1.407	-1.407	-1.407	0.000	20.119	7.66
o50_n16_q3_m8_132	ils-nr	-4.376	-3.336	-2.831	306.542	16.502	11.271
o50_n16_q3_m8_132	ils-nsi	-5.046	-4.599	-4.204	164.916	15.686	36.750
o50_n16_q3_m8_132	ils-nsr	-4.822	-4.501	-4.070	151.585	15.959	37.188
o50_n16_q3_m8_132	ils-os	-3.467	-3.130	-2.737	162.713	17.610	12.039
o50_n16_q3_m8_132	ils-or	-0.396	-0.396	-0.396	0.000	21.351	8.466
o50_n16_q3_m8_211	ils	-4.957	-4.566	-4.324	113.483	11.964	43.67
o50_n16_q3_m8_211	ils-ni	-4.783	-4.393	-4.074	111.499	12.169	33.16
o50_n16_q3_m8_211	ils-ns	-0.939	-0.939	-0.939	0.000	16.698	6.89
o50_n16_q3_m8_211	ils-nr	-4.854	-3.779	-3.305	223.031	12.086	13.432
o50_n16_q3_m8_211	ils-nsi	-4.668	-4.432	-4.219	72.437	12.305	41.121
o50_n16_q3_m8_211	ils-nsr	-4.808	-4.412	-3.938	140.433	12.140	41.679
o50_n16_q3_m8_211	ils-os	-3.710	-3.142	-2.573	202.526	13.433	14.374
o50_n16_q3_m8_211	ils-or	-0.939	-0.939	-0.939	0.000	16.698	7.528
o50_n16_q3_m8_212	ils	-9.940	-9.601	-9.287	129.294	19.713	49.55
o50_n16_q3_m8_212	ils-ni	-9.781	-9.505	-9.295	114.855	19.924	40.55
o50_n16_q3_m8_212	ils-ns	-3.914	-3.914	-3.914	0.000	27.722	8.25
o50_n16_q3_m8_212	ils-nr	-9.119	-8.617	-7.855	254.471	20.804	12.662
o50_n16_q3_m8_212	ils-nsi	-10.293	-9.718	-9.227	195.837	19.243	46.818
o50_n16_q3_m8_212	ils-nsr	-9.967	-9.680	-9.421	94.826	19.677	46.563
o50_n16_q3_m8_212	ils-os	-9.527	-8.689	-7.938	336.753	20.261	13.632
o50_n16_q3_m8_212	ils-or	-3.914	-3.914	-3.914	0.000	27.722	7.972
o50_n16_q3_m8_221	ils	-5.368	-4.980	-4.742	82.884	8.221	41.64
o50_n16_q3_m8_221	ils-ni	-5.368	-4.878	-4.669	71.949	8.221	29.50
o50_n16_q3_m8_221	ils-ns	-1.131	-1.131	-1.131	0.000	13.067	7.81
o50_n16_q3_m8_221	ils-nr	-4.514	-4.033	-3.744	105.938	9.198	10.625
o50_n16_q3_m8_221	ils-nsi	-5.303	-5.088	-4.852	53.957	8.295	39.345
o50_n16_q3_m8_221	ils-nsr	-5.495	-5.146	-4.646	83.507	8.076	38.835
o50_n16_q3_m8_221	ils-os	-4.170	-3.807	-3.425	79.192	9.592	11.177

Continued on next page

Instance	Method	RPD^-	$\overline{RPD}$	RPD^+	σ_{RPD}	RPD^-_{LB}	$\overline{Time}$
o50_n16_q3_m8_221	ils-or	-1.181	-1.181	-1.181	0.000	13.009	7.587
o50_n16_q3_m8_222	ils	-7.461	-7.219	-6.823	119.918	15.205	42.56
o50_n16_q3_m8_222	ils-ni	-7.389	-6.958	-6.661	127.207	15.296	29.09
o50_n16_q3_m8_222	ils-ns	-1.959	-1.959	-1.959	0.000	22.055	7.22
o50_n16_q3_m8_222	ils-nr	-6.307	-5.661	-5.104	187.633	16.642	10.713
o50_n16_q3_m8_222	ils-nsi	-7.578	-7.154	-6.147	227.699	15.060	40.133
o50_n16_q3_m8_222	ils-nsr	-7.597	-7.201	-6.989	124.603	15.036	40.407
o50_n16_q3_m8_222	ils-os	-6.276	-5.459	-5.018	197.692	16.681	11.245
o50_n16_q3_m8_222	ils-or	-3.366	-3.366	-3.366	0.000	20.304	7.834
o50_n16_q3_m8_231	ils	-2.798	-2.252	-1.818	154.825	3.081	48.65
o50_n16_q3_m8_231	ils-ni	-2.461	-2.230	-1.930	83.476	3.438	41.48
o50_n16_q3_m8_231	ils-ns	2.348	2.348	2.348	0.000	8.538	7.61
o50_n16_q3_m8_231	ils-nr	-1.805	-1.558	-1.291	94.792	4.134	13.125
o50_n16_q3_m8_231	ils-nsi	-2.230	-2.138	-2.036	31.214	3.683	46.475
o50_n16_q3_m8_231	ils-nsr	-2.574	-2.278	-2.126	83.361	3.318	47.264
o50_n16_q3_m8_231	ils-os	-1.951	-1.533	-1.036	153.285	3.979	13.975
o50_n16_q3_m8_231	ils-or	1.045	1.045	1.045	0.000	7.155	7.453
o50_n16_q3_m8_232	ils	-2.020	-1.799	-1.586	95.867	7.572	29.66
o50_n16_q3_m8_232	ils-ni	-2.232	-1.892	-1.424	125.565	7.339	24.59
o50_n16_q3_m8_232	ils-ns	1.752	1.752	1.752	0.000	11.713	7.71
o50_n16_q3_m8_232	ils-nr	0.694	1.443	1.947	268.860	10.551	7.405
o50_n16_q3_m8_232	ils-nsi	-2.102	-1.837	-1.365	149.459	7.483	27.476
o50_n16_q3_m8_232	ils-nsr	-2.059	-1.866	-1.554	98.334	7.530	26.743
o50_n16_q3_m8_232	ils-os	1.058	1.458	2.044	226.008	10.951	7.655
o50_n16_q3_m8_232	ils-or	2.024	2.024	2.024	0.000	12.012	8.406