UNIVERSIDADE FEDERAL DE LAVRAS

VITOR NOTINI PONTES

ALGORITHMS FOR THE MULTIPERIOD WORKFORCE SCHEDULING AND ROUTING PROBLEM WITH DEPENDENT TASKS

LAVRAS – MG

2020

VITOR NOTINI PONTES

ALGORITHMS FOR THE MULTIPERIOD WORKFORCE SCHEDULING AND ROUTING PROBLEM WITH DEPENDENT TASKS

Manuscript presented as phase of Defense Exam of the Graduate Program in Computer Science, to obtain the Master's degree.

Prof. DSc. Dilson Lucas Pereira
Orientador

LAVRAS – MG
2020

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Pontes, Vitor Notini.
Algorithms for the Multiperiod Workforce Scheduling and Routing Problem with Dependent Tasks / Vitor Notini Pontes. – Lavras : UFLA, 2020.
89 p. : il.

Dissertação(mestrado acadêmico) –Universidade Federal de Lavras, 2020.
Orientador: Prof. DSc. Dilson Lucas Pereira.
Bibliografia.

1.Combinatorial Optimization.2.Hybrid Exact-Heuristic.3.Workforce Scheduling and Routing Problem.I.Pereira, Dilson Lucas.II.Título.

VITOR NOTINI PONTES

ALGORITHMS FOR THE MULTIPERIOD WORKFORCE SCHEDULING AND ROUTING PROBLEM WITH DEPENDENT TASKS

Manuscript presented as phase of Defense Exam of the Graduate Program in Computer Science, to obtain the Master's degree.

APROVADA em November 3, 2020.

LAVRAS – MG
2020

ACKNOWLEDGEMENTS

First and foremost I would like to thank professor Dilson Lucas Pereira for the valuable teachings and guidance provided throughout the course of this study. Your willingness to help and provide knowledge at any time is inspiring. Having the chance to work with you was an immense pleasure and an experience I will always cherish. I would also like to thank the DCC faculty and staff for providing a healthy learning environment for the students. A special thanks I would also like to direct to professors André Vital Saúde, Mayron César de Oliveira Moreira, and Tiago de Oliveira Januario, who have followed this study since the qualification period, and provided significant input that largely improved the final version of this document. Finally, I would like to thank my life partner, Natana, my parents, Aulus and Jaqueline, and brother, Felipe, for being there for me whenever necessary.

The authors would also like to thank the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 for partially financing this work.

ABSTRACT

Logistics problems lie at the core of industries' everyday operations. These types of problems are also one of the main points of interest in the Operations Research field of study. In recent decades, a class of combinatorial optimization problems, named Workforce Scheduling and Routing Problems (WSRP), has gained significant attention from researchers. In this document, a recently proposed WSRP is studied: in the Multiperiod Workforce Scheduling and Routing Problem with Dependent Tasks (MWSRPDT) a given company provides services by means of mobile teams. Service requests are known beforehand and each service is composed of one or more activities, where one activity may depend on the completion of another. A feasible schedule must be provided where all requested activities are completed in the minimum amount of days while respecting existing dependencies. Two hybrid exact-heuristic approaches based on a Mixed-Integer Programming formulation are proposed. Several new upper bounds for a set of instances obtained from the literature are discovered.

Keywords: Combinatorial Optimization. Hybrid Exact-Heuristic. Workforce Scheduling and Routing Problem.

RESUMO

Problemas de logística se encontram no núcleo das operações de dia-a-dia de diversas indústrias. Estes tipos de problema são também um dos principais pontos de interesse da área de pesquisa conhecida como Pesquisa Operacional. Em décadas recentes, uma classe de problemas de otimização combinatória, chamada Workforce Scheduling and Routing Problems (WSRP), tem recebido significativa atenção dos pesquisadores. Neste documento, um WSRP proposto recentemente é estudado: no Multiperiod Workforce Scheduling and Routing Problem with Dependent Tasks (MWSRPDT), uma empresa oferece serviços através de equipes móveis. Pedidos são conhecidos a priori e cada serviço é composto de uma ou mais atividades, onde uma pode depender do término de outra para que possa ser iniciada. É necessário construir um planejamento viável que visa a realização de todas as atividades requisitadas na menor quantidade de dias possível, enquanto ainda respeitando as dependências existentes. Dois algoritmos híbridos exato-heurísticos baseados em um modelo de Programação Mista-Inteira são propostos. Diversos novos limites superiores para um conjunto de instâncias obtidos da literatura foram discobertos.

Palavras-chave: Otimização Combinatória. Algoritmo Híbrido Exato-Heurístico. Problema de Agendamento e Roteamento de Força de Trabalho.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Logistics related problems have always been given great attention by researchers in the field of Operational Research. Ever since the proposal of the Traveling Salesman Problem (TSP) a wide range of similar problems have emerged and a significant effort has been put by researchers into solving these problems to optimality.

Problems involving the scheduling of personnel to perform activities can be found in the literature dating back to the 70's (MILLER, 1976). For the most part of this period, however, these problems did not incorporate the need for personnel visits to customers. Forward to the end of the century, supported by an increase in urban mobility seen at the time (JONES, 2014), this aspect started being introduced in the personnel scheduling literature.

Since then, the term Workforce Scheduling and Routing Problem (WSRP) has emerged (CASTILLO-SALAZAR; LANDA-SILVA; QU, 2016) as a generalization for problems that deal with the *scheduling* of activities for available workers in a context where there are more requested activities than available personnel. This scenario leads to a need to *route* the teams' visits to the locations where these activities must be performed.

These two aspects have been studied throughout the years in two separate problems that are key to the understanding of this document: The Job Shop Problem (JSP) and the Vehicle Routing Problem (VRP).

1.1 Problem definition

While an extensive amount of research is available on a wide range of Job Shop and Vehicle Routing Problems, the goal of this project is to propose methods for solving a new WSRP, recently proposed in Pereira, Alves and Moreira (2020), named the Multiperiod Workforce Scheduling and Routing Problem with Dependent Tasks (MWSRPDT).

The problem is defined as follows: a given company provides services to customers through mobile teams. This company has a set K of teams and provides a set S of services to its customers. Each service $s \in S$ is a digraph $s = (V_s, A_s)$ in which V_s is a set of tasks that compose the service, and A_s is a set of dependencies between tasks, i.e the existence of an arc $(a,b) \in A_s$ determines that a task a can only be performed after the execution of task b. For simplicity, let us use $a \in s$ meaning $a \in V_s$ and the letters a and b as references to vertices in V_s.

Customers locations are modeled on a complete graph $G = (V,E), |V| = n$. A special vertex, denoted by 1, represents a depot from where each team will depart at the start and arrive at the end of each period, while every other vertex represents customers to be serviced. Vertices in V shall be referenced by the letters i and j.

Each customer i requires the fulfillment of a service $s^i \in S$. We consider a planning horizon for the delivery of services consisting of a set of $1,...,H$ periods. The set of tasks that compose the requested service must be executed at a given point of the planning horizon, respecting existing dependencies.

Each team has an amount $T \in \mathbb{R}_+$ of time available in each period. To move between any pair of vertices i and j, each team spends a predefined time $t_{ij} = t_{ji} \in \mathbb{R}_+$. To execute a task a in a customer i, the time needed by a team k is defined as $t^k_{ia} \in \mathbb{R}_+$.

The objective function is defined by finding an itinerary that minimizes the number of periods required to accomplish every customer's requested services.

1.2 Applicability and Motivation

In this section, we provide the reader insight into our motivation for the writing of this document and present applications of the problem.

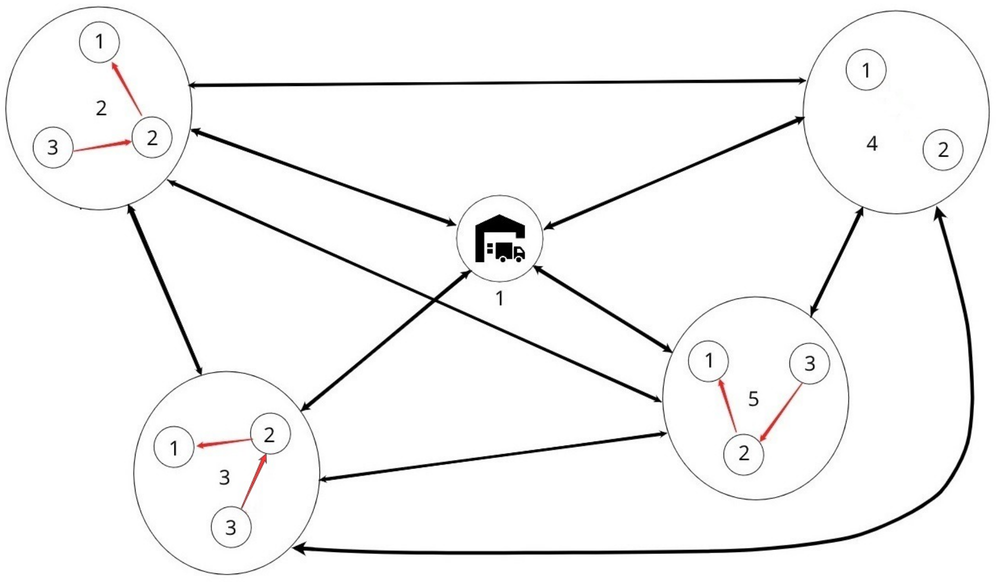
1.2.1 Applicability

Operations Research problems are commonly found in both academic and enterprise environments and are often tied to real-world scenarios, such as automation, scheduling, routing, facility location, among others.

The problem being studied here is no different. It is modeled after a real-world situation found in many service-providing fields that deal with the scheduling of customer visits for available teams of technicians. Good examples of this scenario can be seen in the telecommunication installing, cleaning, gardening, maintenance, and home care fields. In those, a governing unit needs to provide feasible, high-quality schedules for each team/technician, to minimize the number of periods taken to complete every task while ensuring existing constraints are respected.

Consider as an example a cable television provider company. This company offers two services: cable installing and removal. For simplicity, let us refer to these services as S^1 and S^2, respectively. In order to perform S^1 on a new location, three tasks must be performed. First, site availability must be ensured (1). Following, if possible, cables must be installed at the customers home (2). Finally, any modem/router device must be configured (3). The second service, on the

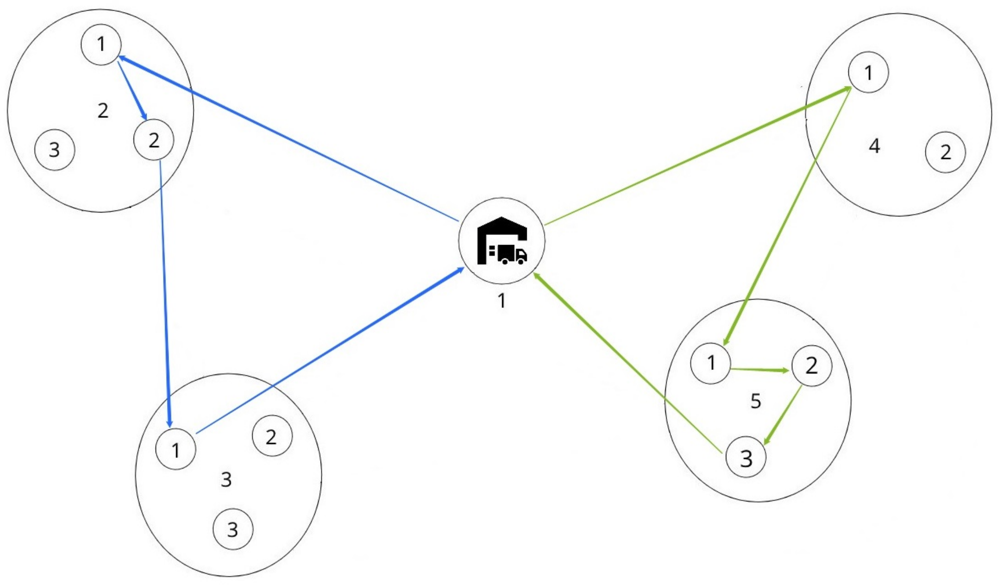
Figure 1.1 – Example of a G' graph for a MWSRPDT instance with 4 customers. The black arrows represent available paths between customers on the complete graph. Red arrows denote dependencies between tasks.



Source: The authors (2020).

other hand, requires only two tasks. Cables must be uninstalled from the customers location (1) and any equipment must be retrieved from their home (2). The existence of precedence between tasks in Service S^1 is inherent in this scenario. Installing cables before checking site availability is not sensible. Also, configuring any device before cables are installed at the requested location is, generally, not possible. Thus, dependency arcs $(2,1)$ and $(3,2)$ exist. Arc $(3,1)$ would also exist, but since it is redundant, it will be omitted throughout this example. In S^2, it makes no difference whatsoever which tasks is accomplished first. Therefore, no dependencies are present. The problem is modeled on an extended complete graph $G' = (V', E')$. Let us define $V' = \{\{2,1\},\{2,2\},\{2,3\},\{3,1\},\{3,2\},\{3,3\},\{4,1\},\{4,2\},\{5,1\},\{5,2\},\{5,3\}\}$ as the set of every requested (customer, task) pair. A visual representation of G' is presented in Figure 1.1. The centered vertex 1 marks the depot, where all teams leave and must return to at the end of the workday. In our example, $K = 2$ teams are available and 4 customers are present. Large circles represent customer sites. Inside each large circle, small circles represent the tasks that amount to the service requested by each customer, i.e, the nodes in V'. Red arcs between tasks denote dependencies between tasks. In this case, customers 2, 3 and 5 requested Service S^1. Service S^2 is requested only by Customer 4.

Figure 1.2 – Solution for the first day of an MWSRPDT example instance.



Source: The authors (2020).

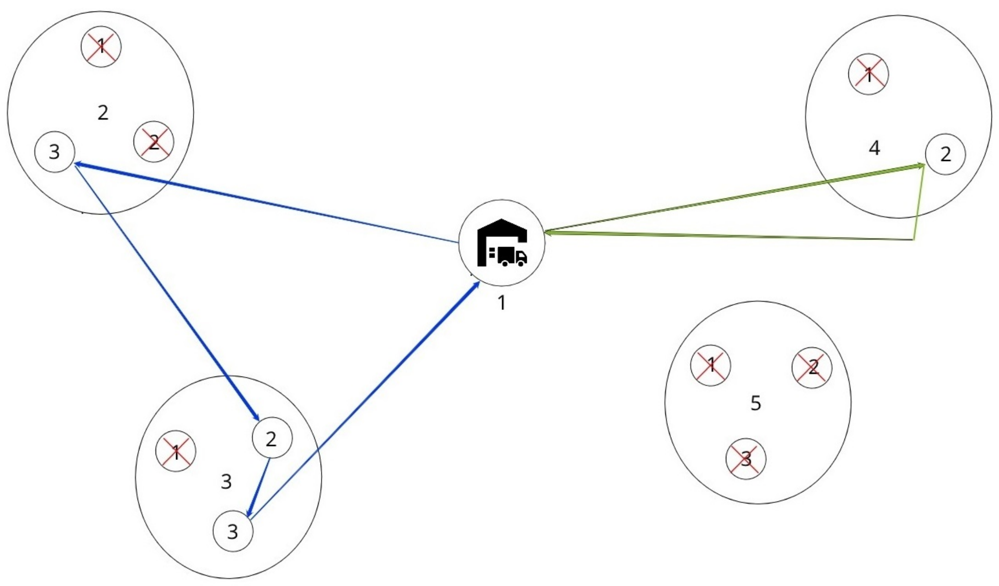
An example solution to the problem can be seen in Figures 1.2 and 1.3. On the first day, Team 1, whose route is represented by the blue color, visits Customer 2 and performs tasks $\{2,1\}$ and $\{2,2\}$, respectively. Once task $\{2,2\}$ is finished, task $\{2,3\}$ becomes available. However, the route planner decided it would be best to visit customer 3 and perform task $\{3,1\}$. After task $\{3,1\}$ is finished, Team 1 returns to the depot and finished its workday. Team 2, represented by the green color, also visits two customers on the first day. After leaving the depot, a visit is made to Customer 4, where task $\{4,1\}$ is performed. Following, the team leaves for Customer 5 and performs tasks $\{5,1\}$, $\{5,2\}$ and $\{5,3\}$, respectively, before returning to the depot. After having all its requested tasks completed by Team 2 on the first day, customer 5's request is now finished.

Starting on the second day, Team 1 visits customers 2 and 3 to perform the remaining tasks, returning to the depot after completing task $\{3,3\}$. At the same time, Team 2 leaves for Customer 4 and performs task $\{4,2\}$ before return to the depot. Once both teams arrive, every requested service is completed in a solution that spans 2 periods.

1.2.2 Motivation

Increasing the effectiveness of logistics and operations will always be a key aspect of a company's success. The ability to better distribute available resources, leading to lower opera-

Figure 1.3 – Solution for the second day of a MWSRPDT example instance. Tasks completed on previous days are crossed in red.



Source: The authors (2020).

tions costs, will often lead to better results for the business, be those financial or organizational. Hence, problems like the Job Shop Problem and the Vehicle Routing Problem have been formulated many decades ago in an attempt to generalize these scenarios and, currently, continue to be extremely relevant.

Although we are dealing with a new problem, similar problems can be found in the everyday operations of a fair amount of companies and also in the literature. So far, hybrid exact-heuristic approaches to solving a problem with the singularities presented in the MWSRPDT are yet to be seen in the literature.

1.3 Workforce Scheduling and Routing Problems

Workforce Scheduling and Routing Problems can be described as a class of problems that deal with the dispatching of employees to perform tasks in different locations, where the number of employees is smaller than the number of requested tasks. Hence, resulting in a need to allocate employees to tasks and plan routes that define how these visits, which can be performed by any means of transportation available, will occur.

Within this context, a significant number of different problems can be formulated by adding and/or modifying existing constraints. The most common ones found in the literature involve the inclusion of time windows, fleet heterogeneity, employee-specific skill sets, fleet or employee geographic restrictions, dependencies between tasks, etc. Besides, a very high number of possible objective functions can also be envisioned. These functions range from reducing costs and/or travel time, increasing productivity or minimizing the workdays needed to fulfill every requested task to simply achieving better human resource planning and an efficient internal organization (BOSTEL et al., 2008).

WSRPs have a strong correlation to two well-studied VRP formulations: the Capacitated Vehicle Routing Problem (IRNICH; TOTH; VIGO, 2014) and the Periodic Vehicle Routing Problem (PVRP) (IRNICH; SCHNEIDER; VIGO, 2014). The former, since employees have, typically, previously known skill sets with predefined times for the completion of tasks. The latter, because all requested tasks not often can be completed in a single days' schedule. Thus, a multiperiod scenario is required.

1.4 Related Problems

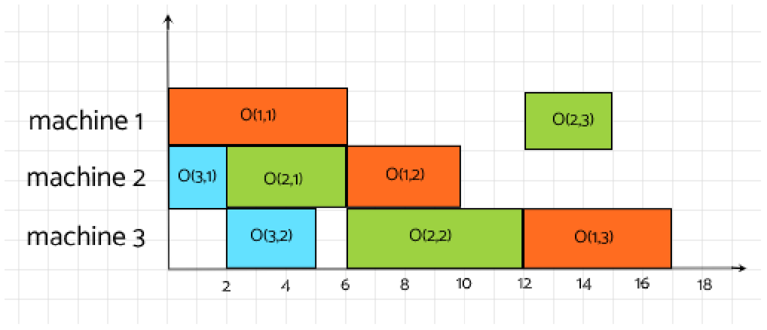
Two classic Operations Research literature problems are intimately related to the proposed problem: the Flexible Job Shop Problem and the Vehicle Routing Problem. Both of these problems are known to be NP-hard (LENSTRA; KAN; BRUCKER, 1977) (LENSTRA; KAN, 1981), a fact that allows the understanding that both key aspects found in our proposed problem are also NP-hard.

In this section, the reader will find a brief discussion about both these problems and how they relate to our own.

1.4.1 The Flexible Job Shop Problem

The Job Shop Problem (JSP), also known as Job Shop Scheduling, is among the most classical optimization problems studied to date. In the JSP, we are given a set M of m machines and a set J of n jobs. Each job consists of a set of o_j operations $O_j = (j,1),...,(j,o_j)$. Each operation (j,i) has to be executed on a machine $M(j,i)$, where it takes time $t(j,i)$ to be executed. Operations in O_j should be executed in order. One possible objective is to minimize the makespan, defined as the moment the last job is complete. In Figure 1.4 we provide a visual representation of a schedule for an example JSP instance with 3 jobs and 3 machine, with a makespan of 17 time units.

Figure 1.4 – Example of a Job Shop Problem final schedule for a 3x3 instance.



Source: The authors (2020).

The Flexible Job Shop Problem (FJSP) is a generalization upon the original JSP that increases its difficulty by allowing any machine to perform any operation, where every machine can (although not obligatory) have different execution times for a given operation. Due to its inherent logistical characteristics, it is often related to real-world situations commonly found in the industry and so, it has been given great attention by researchers since the 1960's.

The FJSP concept heavily relates to our proposed problem, since we also deal with a similar assignment problem (*which* team/machine will perform an task/operation) and a similar scheduling problem (*when* will the assigned team perform such task).

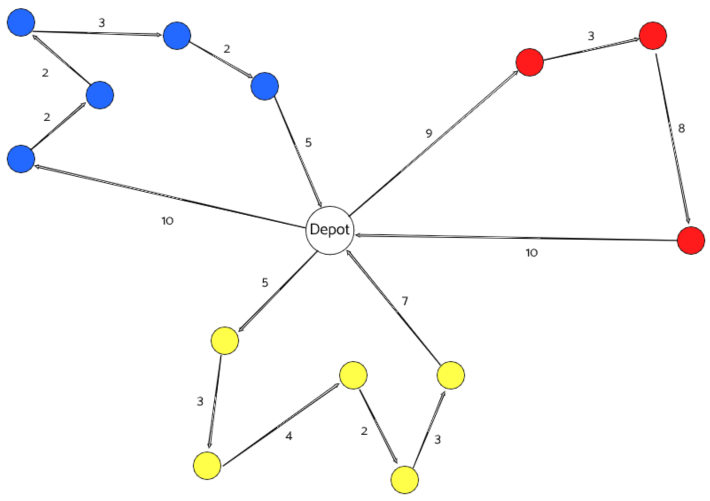
1.4.2 The Vehicle Routing Problem

First introduced by Dantzig and Ramser (1959), the Vehicle Routing Problem is, perhaps, the most studied Operations Research problem to date. It generalizes the commonly known TSP and can be described as the problem of designing optimal delivery or collection routes from one or several depots to a number of geographically scattered cities or customers, subject to side constraints (LAPORTE, 1992).

In a classical VRP solution, a set of one or more vehicles must leave the depot, perform a route visiting customers and return to the depot at the end of its route. Most common objective functions for a VRP revolve around minimizing the cost of fulfilling every customer's requests or minimizing the total distance travelled. In Figure 1.5, we provide an example of a classic VRP: each circle is a customer that needs visiting and the value in the edge between any pair of customers is the cost of traveling that edge. In the provided example, three routes were designed to service every customer.

Variations upon the original VRP are often built to simulate real-world scenarios since vehicle logistics have become a matter of great importance in today's world. The proposed

Figure 1.5 – Example of a VRP with 13 customers to be visited.



Source: The authors (2020).

problem incorporates the key aspect of the VRP in the sense that optimal routes have to be designed for each team to perform its assigned tasks. The MWSRPDT is strongly related to a class of notoriously difficult to solve VRPs (VIDAL; LAPORTE; MATL, 2020), named Vehicle Routing Problem with Multiple Synchronization Constraints (VRPMS). In these, a change applied to one route can have significant impact on other routes. In the worst case, rendering the whole set of routes infeasible (DREXL, 2012).

1.4.3 Document Structure

This document is structured as follows: Section 2 provides a literature review of a set of works in the field and the solution methods behind them. In Section 3 we present a constructive heuristic and two hybrid exact-heuristic algorithms developed for solving the MWSRPDT. In Section 4 we introduce the problem instances used in this document followed by a discussion regarding the results found by our algorithms for those. In Section 5 we discuss some of the developed approaches while writing this document that were eventually discarded. Finally, in Section 6 we present our concluding arguments and direction of future research.

2 LITERATURE REVIEW

2.1 Problem Overview

Due to the connection of the problems discussed in this document with real-world logistics concepts, variations of the original problems are constantly emerging in the literature in attempts to incorporate new constraints originated from particular contexts. Nevertheless, the manner in which researchers try to solve these problems seldom differs from one another by a large margin. In this section, the reader will find an overview of papers that are both important to the field of research and also related to our problem.

2.2 Solution Methods

Optimization problems are typically approached by three different types of solution methods: exact, heuristic and hybrid methods. Exact methods will attempt to solve a problem and prove solution optimality. These methods are, usually, very computationally expensive. For that reason, heuristic approaches are often used due to their ability to deliver relatively fast solutions. On the other hand, the quality of these solutions seldom matches the ones found by exact methods. In recent years, hybrid methods that attempt to utilize both concepts in conjunction have emerged and have shown promising results.

2.2.1 Exact Methods

Solutions for NP-hard optimization problems generated by exact methods are usually obtained from the *branch-and-bound* algorithm. The *branch-and-bound* algorithm is, in essence, an enumerative scheme for solving optimization problems that diminishes the search space analyzed for a solution by using the concepts of *lower* and *upper* bounds (MITTEN, 1970). Let z be the optimal solution value for a given minimization problem. An upper bound for z is defined by any value $\bar{z} \geq z$ and a lower bound is defined by any value $\underline{z} \leq z$. Using the values provided by both bounds, the algorithm can simply ignore portions of the solution space where the value of the solutions found is not sandwiched between the upper and lower bounds, therefore speeding the process of finding a solution.

However, finding *good* bounds are is an easy task and the quality of the bounds found is strongly tied to the algorithms' ability to provide a solution in a feasible time. Upper bounds are usually found by heuristic methods, while lower bounds are usually found by Mixed Integer Linear Programming (MILP) formulations. When dealing with minimization problems, in general, finding good lower bounds is usually a more difficult task than finding good upper bounds.

A MILP formulation describes the programming through sets of linear equations where some variables can only assume *integer* values. Given a formulation, obtaining a lower bound involves solving a simplified version of the problem, usually named *relaxation*. Since a problem can be formulated in various ways, the quality of the bounds provided by each formulation will follow the strength of the formulation itself. For a detailed guide on mathematical formulation methods for optimization problems, we refer the reader to (WOLSEY, 1998).

2.2.2 Heuristic Methods

Heuristic methods are designed to deliver fast but not guaranteed optimal solutions for optimization problems. Ranging from very simple to very complex algorithms, heuristics have been applied to all sorts of optimization problems throughout the years.

The vast majority of works tackling the FJSP in the literature employ some sort of heuristic algorithm (CHAUDHRY; KHAN, 2016). In the VRP literature, even though this heuristic algorithm dominance is not so prevalent, heuristic algorithms have been proven capable of reaching less than 1% gaps to the Best Known Solution (BKS) in classical problem instances (VIDAL et al., 2013).

The most important classes of heuristic methods, which we will detail in the following sections, are the *constructive* heuristics, *improvement* heuristics and *metaheuristics*.

2.2.2.1 Constructive Heuristics

Constructive heuristics are usually simplistically designed algorithms. In combinatorial optimization problems, solutions are combinations of pieces that constitute a feasible arrangement. Constructive heuristics start with an empty solution and iteratively add new components, according to some criterion, until a feasible solution is obtained.

As an example, let us consider the TSP, defined as follows: given a set of cities, with paths available between every pair of cities, what is the cheapest route available for starting in a random city, visiting all other cities, and coming back to the original city? A simple algorithm that constructively solves this problem is a greedy concept utilized in many fields: the Nearest Neighbor heuristic for the TSP defines as the next city to be visited the city that can be reached with the smallest traveled distance coming from the current city (GUTIN; YEO; ZVEROVICH, 2002). This process is repeated until there are no more cities to be visited. With this method, a low computational cost solution is provided. However, that solution rarely proves to be a high-quality one.

Most notable constructive heuristics for the VRP date back from the 60s up to the 80s, with the works of Clarke and Wright (1964), Mole and Jameson (1976), Fisher and Jaikumar (1981) and Solomon (1987), among others. These pioneer works have been heavily used in the following decades and, to this day, remain relevant when dealing with many different variants of VRPs.

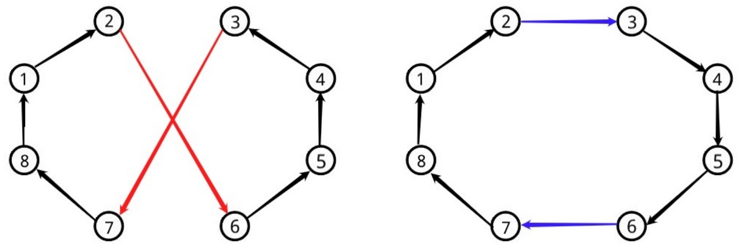
2.2.2.2 Improvement Heuristics

Differently from constructive heuristics, improvement heuristics derive a new solution based on an already complete solution. First, let us establish the concept of a solution *neighborhood* when dealing with discrete optimization problems. As defined by Talbi and El-Ghazali (2009), the neighborhood $N(s)$ of a solution s can be represented by the set $\{s' : d(s', s) \leq \varepsilon\}$, where d is a function that yields the distance between two solutions s' and s after applying a *transformation* to s.

The concept of distance will vary based on the encoding used to represent a solution to the problem at hand. For example, given a binary representation for a solution s to a discrete optimization problem, its *neighborhood* $N(s)$ is usually defined based on the Hamming distance. Let d be the Hamming distance and $\varepsilon = 1$. A *neighbor* solution $s' \in N(s)$ of s is defined by any solution that can be reached by flipping one bit of s, where the flip of a bit is a well-defined type of transformation.

A common type of transformation applied to a TSP solution is the 2-opt algorithm (CROES, 1958). The 2-opt algorithm works by analysing neighbours obtained by removing two nonadjacent edges in a solution and reconnecting affected vertices in a different way, resulting in a part of the solution becoming reversed. Figure 2.1 provides an example of a 2-opt move. In the initial route $1 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 7 \rightarrow 8 \rightarrow 1$, edges $(2,6)$ and $(3,7)$ are broken and the reconnected in a different way, providing the new solution $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$ $\rightarrow 1$. The 2-opt neighborhood of a solution is the set of all new solutions made possible by applying 2-opt moves to the original solution.

Figure 2.1 – Example of a 2-opt swap.



Source: The authors (2020).

The most common framework for the application of improvement heuristics is Local Search (LS). Let s be a reference solution to a minimization discrete optimization problem and $f(s)$ be a fitness function that evaluates the quality of a solution. Given a well-defined neighborhood function $N(s)$, the Local Search algorithm travels through the candidate solutions $s' \in N(s)$ evaluating every candidate solutions fitness value $f(s')$. A candidate solution $s' \in N(s)$ with fitness value less than all its neighbours is called the *local optimum* of $N(s)$. Let s^* be the local optimum of a neighborhood $N(s)$. If $f(s^*) < f(s)$, s^* becomes the new reference solution. This process is repeated until reaching a solution s^* that provides no improvement upon the current reference solution or predefined stoppage criteria has been reached.

The quality of a solution found by a LS algorithm is tightly bound by the quality of the neighborhood function used concerning the problem at hand. Problem characteristics, as well as solution representation, play an important role in the effectiveness of a neighborhood. The same 2-opt neighborhood, when applied to a scheduling problem like the FJSP, will generally not produce high-quality solutions.

In order to escape the limitations of LS-based algorithms, more elaborate algorithmic schemes, named *Metaheuristics*, have been developed. In the following section, we provide further insight into this class of algorithms.

2.2.2.3 Metaheuristics

As defined by Glover and Kochenberger (2003), metaheuristics are solution methods that orchestrate an interaction between local improvement procedures and higher-level strategies to create a process able to escape from local optima and perform a robust traverse of the search space. These solution methods work as frameworks that can be applied to a wide range of optimization problems, always aiming for a more thorough exploration of the solution space.

This broader exploration often leads to the discovery of a solution that better satisfies the problems objective function.

Metaheuristics can be divided into two groups according to their exploratory characteristics: single-solution based and population-based metaheuristics. The first uses an initial solution as a guideline and tries to explore its neighborhood in search of a better solution, reiterating that process until predefined criteria are met. The most common examples of single-solution based metaheuristics include Simulated Annealing (KIRKPATRICK; GELATT; VECCHI, 1983), Tabu Search (GLOVER, 1989), Variable Neighborhood Search (MLADENOVIć; HANSEN, 1997) and Adaptive Large Neighborhood Search (ROPKE; PISINGER, 2006). Population-based metaheuristics act differently by attempting to create new solutions by combining aspects of already discovered solutions. Among these methods are Genetic Algorithms (HOLLAND, 1992), Particle Swarm Optimization (KENNEDY; EBERHART, 1995) and Ant-Colony Optimization (DORIGO; COLORNI; MANIEZZO, 1991) metaheuristics.

The use of metaheuristics has shown significant increase in the last decades, largely due to their adaptability and ability to achieve a good balance between solution quality and computational effort.

2.2.3 Hybrid Solution Methods

In recent years researchers have efforted in combining the exploratory characteristics of different solution methods as well as their interactions in attempts to design new *hybrid* algorithms. These solution methods will often build a solution by solving different aspects of the original problem separately and then combining those or by orchestrating the use of different solution methods in parallel.

Comparing the quality of 64 heuristic approaches against 20 benchmark CVRP instances, the work of Vidal et al. (2013) provides an interesting insight for future research. Their study shows that the best performances are achieved by hybrid methods combining neighborhood-centered search with collective intelligence and population concepts as well as parallel cooperative methods that combine solution-recombination procedures with Tabu Search. These results can be viewed as supporting arguments for the recent trend in hybridization seen in the literature.

The combination of exact and heuristic methods for the creation of new hybrid algorithms has also been in the literature for a few decades, with notable works such as Local

Branching (FISCHETTI; LODI, 2003). The Local Branching framework makes use of a MIP solver as an exploratory device to travel the neighborhood, defined as the set of every solution that does not differ by more than *k* variables (Hamming distance) for a sufficiently small *k*, of a given reference solution. Algorithms that look for solutions by combining exact and metaheuristic solution methods are often presented under the term Matheuristic (MANIEZZO; STüTZLE; VOSS, 2010). These algorithms are usually applicable when a subproblem of a hard combinatorial optimization problem can be efficiently solved by an exact algorithm. Once this *easy* subproblem is solved to optimality, an heuristic procedure is used to try and build upon this partial solution by looking for a complete one.

Although the proposal of hybrid methods is not new, in recent years it has been given more attention by researchers, with competitive results to a variety of combinatorial optimization problems being found with the combination of exact methods and metaheuristics with improvement heuristics (BLUM et al., 2011).

2.3 Workforce Scheduling and Routing Problems

Although the WSRP literature in itself is not vast, its intrinsic characteristics have been thoroughly studied in its related problems for many decades. Most of the works found in the literature are attached to the technician scheduling and home care application areas. This characteristic leads to a situation where WSRPs are often tied to real-life scenarios, making algorithm performance comparison very difficult, since problem instances are ordinarily case-specific (CASTILLO-SALAZAR; LANDA-SILVA; QU, 2016).

Due to the highly constrained environments that typically appear in most WSRPs, the use of purely exact solutions becomes increasingly difficult. Hence, heuristic and metaheuristic algorithms have often been used in the literature as the chosen solution methods for these types of problems, mostly due to their higher ease for adaptation and good trade-off between solution quality and computational time.

2.3.1 WSRP Literature Review

Most of the common heuristic solution methods for solving Combinatorial Optimization problems have been used to a great extent for solving problems that involve routing and scheduling scenarios. However, when dealing with WSRPs, most of the heuristic proposals seen in the literature come in the form of exploratory heuristics that start from a reference initial solution and make use of swap and insert operators to define solution neighborhoods that are explored in search of better solutions (CORDEAU et al., 2010) (XIE; POTTS; BEKTAŞ, 2017). Although not as often as heuristic approaches, exact (BREDSTRöM; RöNNQVIST, 2007) (ZAMORANO; STOLLETZ, 2017) and hybrid proposals (GOEL; MEISEL, 2013) (ÇAKıRGIL; YüCEL; KUYZU, 2020) can also be found in the literature, mostly making use of Branch-and-Bound and Branch-and-Cut algorithms that are, occasionally, used in combination with exploratory heuristics.

A problem in the context of telecommunications maintenance and installation is tackled by Cordeau et al. (2010). In their proposed problem a set of tasks must be completed by a set of available technicians with heterogeneous skill sets, precedence between activities must be respected, team building can be necessary and outsourcing is a valid option. In their problem, however, travel costs are not considered. A constructive heuristic in which assignments are based on a weighted function of the criticality of the task, the number of technicians and the specific skills it requires is combined with the Adaptive Large Neighborhood Search (ALNS) framework in which destroy and repair operators are selected based on past performance.

In Kovacs et al. (2012) the authors work on a problem derived from the context of infrastructure and maintenance service providers, which they name Service Technician Routing and Scheduling Problem (STRSP). In the STRSP activities must be performed using mobile technicians with heterogeneous skill sets, time windows must be respected and outsourcing is a possibility. Therefore, the objective is to minimize the sum of routing and outsourcing costs. The problem is modeled with two distinct variations: one that requires team building and one that does not. Both are solved using the ALNS metaheuristic in an algorithm that rewards pairs of destroy and repair operators instead of each one by itself, which is the most common approach.

A hybrid approach consisting of a constructive heuristic, improvements by ALNS and a Set-Covering formulation for the TRSP is proposed in Pillac, Guéret and Medaglia (2013). Their problem has a singular characteristic: the existence of available resources in the form of tools (renewable) and spare parts (non-renewable) that are required for the completion of a given activity. Technicians leave and go back to their houses instead of a depot and also start with a predefined set of tools and spare parts with them. If needed, they can visit a depot for replenishment. An initial solution is constructed using a regret-based constructive heuristic, in which the next assigned activity is the one whose regret value (estimation of the additional cost incurred if a request is not inserted at its best position) is the highest among available activities. Destroy and repair operators are designed for improvement in a ALNS framework and afterward, a set-covering problem is solved using the best solutions found in each iteration of the ALNS procedure. The set-covering problem implementation managed to find solutions 1.5% better on average than the ones found by ALNS.

An exact approach for the multiperiod Technician Routing and Scheduling Problem (TRSP) is proposed in Chen, Thomas and Hewitt (2016). In their approach, technicians have learning capabilities, meaning every time a technician performs an activity he will gain experience. The accumulated experience can lead to him performing that same task in less time in the future. Technicians' skill sets are heterogeneous and their learning rates and initial experience are known. Even though a multiperiod horizon is considered, only the current daily demand is known at the beginning of each day and, therefore, future demands are not considered. A mathematical model that seeks the completion of every requested activity in the least amount of time possible and accounts for technicians learning is proposed with routes being designed by what the authors describe as a record-to-record travel algorithm.

A WSRP in a multiperiod environment, similar to ours, is presented in Zamorano and Stolletz (2017), who proposes a Branch-and-Price approach to the Multiperiod Technician Routing and Scheduling Problem (MPTRSP). In their MPTRSP, that differs from the MWS-RPDT due to tasks having time windows that must be respected and employees needing to be grouped into teams, precedence constraints between tasks are also present. The problem is solved by a Branch-and-Price algorithm that is applied to two different problem decompositions: one in which the problem is viewed as multiple single period problems (day decomposition) and another in which every possible team composition is formed and teams are assigned to days in the planning horizon (team-day decomposition), turning the problem into a problem of finding routes for each team on each day. Their algorithm is benchmarked against adapted VRPTW instances commonly seen in the literature where the team-day decomposition proved to be the better option, achieving results far superior to the day decomposition and a regular MIP solver.

A seldom seen but closely related to real-world scenarios characteristic, which the authors name the RASTA problem, is dealt with in Zhan and Wan (2018). In their work, service times are uncertain, being obtained by generating scenarios based on historical data. A scenario-based MIP model is proposed, however, its use was deemed impossible for large-sized problem instances. Hence, a Tabu Search algorithm is presented, managing to reach good quality solutions at a reasonable time for problems with less than 40 customers. This is tied to the fact that the TS algorithm still makes use of the scenario-based model to determine appointment times and routing schedules costs. In Shi, Boudouh and Grunder (2019), uncertainty is also present in travel and service times of workers in a similar home care context. According to the authors, the absence of uncertainty in travel times can lead to very weak scheduling strategies that often lead to delay in service delivery. Hence, uncertainty is incorporated in their work. Three (Simulated Annealing, Tabu Search, Variable Neighborhood Search) heuristic algorithms were employed, with Tabu Search reaching the best overall results.

As we have previously stated, a large number of WSRP works in the literature deal with problems derived from home care scenarios. In Moussavi, Mahdjoub and Grunder (2019), the authors combine a Sequencing Generalized Assignment Problem (SGAP) with the need to route home care workers' visits to clients. A mathematical model that deals with both aspects is presented for the problem. However, the computational time required to solve even small to medium-sized instances is high. For that reason, a three-step matheuristic is employed. By decomposing the problem into three aspects: (i) finding the optimal number of workers required to service every request; (ii) grouping requests into sets and building routes for those and (iii) assigning routes to available workers, the matheuristic can reach optimality in 82% of the problem instances evaluated in significantly less time.

Another important application of WSRPs is on disaster relief scenarios. In Moreno, Munari and Alem (2019) the authors tackle a Crew Scheduling and Routing Problem in road restoration. Aside from the non-trivial characteristics of WSRPs, a degree of difficulty is added in this context due to the number of paths between nodes being dynamic. At given points in the algorithm's execution, nodes in the network might be inaccessible due to road damage. A Branch-and-Benders-Cut algorithm is proposed. Their algorithm, based on the Benders decomposition (BENDERS, 1962), attempts to overcome MIP solvers' inability to solve the original mathematical model for the problem by decomposing the original problem into a Master Problem (MP) with only the complicating variables and two Subproblems (SP). The MP deals with scheduling decisions for the crew, while the SPs deal with the routing of these visits. Feasibility and optimality cuts are added to the problem at every newly encountered integer solution. Their proposed algorithm manged to find optimal solutions to 41.67% of evaluated instances as well as finding valid lower bounds for the first time for all of them.

In Guastaroba, Côté and Coelho (2020), the authors deal with the Multiperiod Workforce Scheduling and Routing Problem (MPWSRP). In the MPWSRP, workers are specialized in their skill domains. However, within their skill domains, workers' proficiency is homogeneous. In order to perform a requested job, workers need to be grouped into teams in a way where workers' skill sets within the team amount to the proficiency level required to perform such a job. Jobs are composed of a single task and no dependencies between them exist. An ALNS and a Decomposition Algorithm, that sequentially solves each aspect of the problem individually, are proposed. The ALNS algorithm outperforms the decomposed approach and manages to compete with a commercial solver on small-sized instances.

Table 2.1 – Feature comparison between WSRP works found in the literature and the MWSRPDT.

	Attributes								
Reference	Multiperiod	T.W.	T.D	Het. S.	Hom. S.	T.B.	Precedence	Outsourcing	Solution Method
(BOSTEL et al., 2008)	✓	✓	✓		✓				Hybrid
(DOHN; KOLIND; CLAUSEN, 2009)		✓	✓	✓		✓			Branch-and-Price
(CORDEAU et al., 2010)	✓	✓		✓		✓	✓	✓	ALNS
(PILLAC; GUéRET; MEDAGLIA, 2011)		✓		✓					ALNS
(KOVACS et al., 2012)		✓	✓	✓		✓		✓	ALNS
(RASMUSSEN et al., 2012)	✓	✓	✓	✓		✓			Branch-and-Price
(CHEN; THOMAS; HEWITT, 2016)	✓	✓	✓	✓					Hybrid
(ZAMORANO; STOLLETZ, 2017)	✓	✓	✓	✓		✓	✓		Branch-and-Price
(ZHAN; WAN, 2018)	✓	✓	✓		✓				Tabu Search
(SHI; BOUDOUH; GRUNDER, 2019)	✓	✓	✓	✓					Metaheuristics
(MOUSSAVI; MAHDJOUB; GRUNDER, 2019)	✓	✓	✓		✓				Matheuristic
(MORENO; MUNARI; ALEM, 2019)			✓		✓				Branch-and-Benders-Cut
(GUASTAROBA; CôTé; COELHO, 2020)	✓		✓		✓	✓			ALNS
(ÇAKıRGIL; YüCEL; KUYZU, 2020)		✓	✓	✓		✓		✓	Matheuristic
MWSRPDT	✓		✓	✓			✓		Hybrid

Legend: T.W. - Time-Windows | T.D. - Travel distance | Het. S. - Heterogeneous Skills | Hom. S. - Homogenenous Skills | T.B. - Team Building

Source: The authors (2020)

Although many features are common among many of the WSRP variants studied in the literature, the existence of precedence constraints is rare, as shown in Table 2.1. In spite of adding significant complexity to the problem, these can occur in many sectors and, therefore, should be a vital aspect considered for further WSRP research (Khalfay; Crispin; Crockett, 2017).

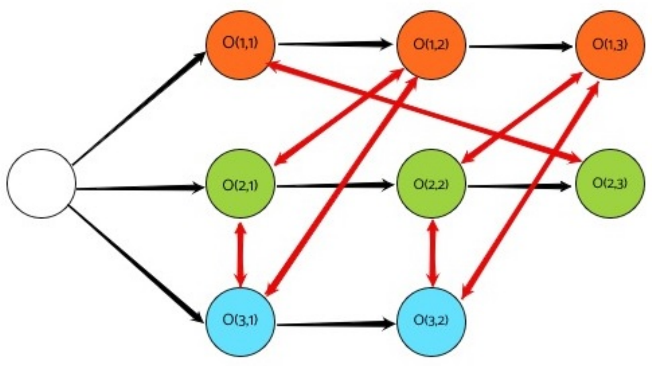
2.4 The Flexible Job Shop Problem

A typical JSP instance is represented as two matrices A and B, where each row i is a job and each column j is an operation of job i. The value in $A_{i,j}$ is the name/index of the machine that is required to perform operation (i, j) and the value in $B_{i,j}$ corresponds to the amount of time machine $A_{i,j}$ needs to perform operation (i, j). An example graph representation of the 3x3 instance shown in Figure 1.4 is provided in Figure 2.1.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 2 & 3 & 0 \end{bmatrix} B = \begin{bmatrix} 6 & 4 & 5 \\ 4 & 6 & 3 \\ 2 & 3 & 0 \end{bmatrix} \tag{2.1}$$

The problem is usually visualized as a disjunctive graph, as exemplified by Figure 2.2. Each vertex represents an operation and each operation is dependent on the completion of its predecessor, as shown by the black arcs between any pair of vertices. The red arcs between any pair of operations denote that both are to be performed on the same machine. The disjunctive graph is useful in developing algorithms for the JSP, as a solution for the problem corresponds to finding orientation for the red arcs such that no cycles remain in the graph.

Figure 2.2 – Graph representation of the 3 job-sized instance defined in 2.1.



Source: The authors (2020)

A common variation of the JSP is the Flexible Job Shop Problem (FJSP), in which there is no previous assignment of operations to machines, i.e any machine can perform any operation and each machine can have a different completion time for every operation. The FJSP can be seen as two separate sub-problems: a Routing problem and a Scheduling problem. Initially, there is the decision of how to assign each operation to a machine, which we will call the Routing subproblem. Then, what remains is a Scheduling subproblem, where the set of operations assigned to each machine has to be scheduled in a sequential order attempting to minimize a given objective function, respecting existing precedence constraints. As displayed by Figure 2.3, in the FJSP, the red arcs between any pair of vertices *initially* do not exist. The *routing* of operations to machines is also part of the problem.

Given the problem definition, a large number of objective functions can be defined. The most common ones found in the literature, mostly due to the natural characteristics of scheduling problems trying to simulate real-world situations, are the minimization of the total makespan, i.e the total completion time of the set of operations, and the minimization of total tardiness (schedule delay).

Let C_i be the completion time of each job i in a problem with N jobs. The minimal makespan function is defined as:

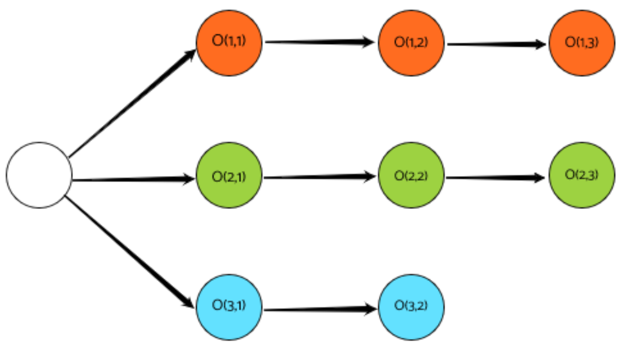
$$\min\left(\max_{i=1,\ldots,N} C_i\right). \tag{2.2}$$

Minimal total weighted tardiness is given by:

$$\min\sum_{i=1}^{N} v_i T_i, \tag{2.3}$$

where the tardiness $T_i = \max(0, C_i - d_i)$ is the amount by which the completion time exceeds the due date d_i and the weight v_i measures the priority of jobs.

Figure 2.3 – Graph representation of a 3 job-sized Flexible Job Shop instance before machine assignment.



Source: The authors (2020)

Many different exact and heuristic approaches for the solution of scheduling problems can be found in the literature dating back to the 1960s, with most of the works presenting Integer Programming formulations for the problem. According to Ku and Beck (2016), when dealing with exact methods, the time-indexed formulation, the rank-based formulation and the disjunctive formulation of Bowman (1959), Wagner (1959), and Manne (1960), respectively, are still widely used when solving scheduling problems.

State-of-the-art heuristic solutions are known to reach 0.5 to 1.0% distance to optimal solutions for some variations of scheduling problems. These methods can be adapted to any slight change on the original problem with a lot more ease than exact methods, as well as having significantly lower running times than those. Therefore, whenever proof of optimality is not a requirement, their use is often advantageous.

While the choice of the heuristic used for the proposed algorithm varies, the manner in which most constructive algorithms found in the literature work is quite similar. A set of one or more dispatching rules is defined and based on these rules, an initial solution is designed. From this initial solution, local search or similar exploration methods are used to try and improve such solution, with the intent of achieving an improved final solution that better satisfies the objective function.

Dispatching rules can be described as the sequencing strategy employed to prioritize task scheduling. The importance of a good initial solution and the choice of a good dispatching rule are key aspects to achieving an optimal or near-optimal final solution, as made evident on the results obtained by Brandimarte (1993), who compares results obtained with three different dispatching rules, coupled with a Tabu Search inspired algorithm.

Due to their proven importance, finding a high-quality dispatching rule for scheduling problems has become a field of study in its own right with works ranging from surveys and comparisons (BLACKSTONE; PHILLIPS; HOGG, 1981) (CHANG; SUEYOSHI; SULLIVAN, 1996) (BRANKE; HILDEBRANDT; SCHOLZ-REITER, 2015) to works focused on proposing new effective rules (HOLTHAUS; RAJENDRAN, 1997) (DOMINIC; KALIYAMOORTHY; KUMAR, 2004) and even algorithms and linear programming models designed to find the best suitable dispatching rules for a given scheduling problem (LI; OLAFSSON, 2005) (TAY; HO, 2008) (NGUYEN; ZHANG; TAN, 2016). More so in the last decade, several studies (INGIMUNDARDOTTIR; RUNARSSON, 2018) (PRIORE et al., 2018) (JUN; LEE; CHUN, 2019) using data science techniques dedicated to discovering dispatching rules have been presented. The use of these techniques can be advantageous due to being able to use problem-specific data in the discovery process.

In an attempt to find the best dispatching rule in an environment with more than one objective function (multi-criteria), Amin and El-Bouri (2018) propose a minimax Linear Programming (LP) model to find a dispatching rule that suits moderately well all of the available criteria. Making use of 26 different dispatching rules found in the literature as well as a self-introduced rule, the authors convert the problem into a preference voting system that defines a *desirability* equation that provides a score for every dispatching rule according to every available criterion. The LP model is then solved by maximizing the *desirability* score of every dispatching rule.

When employing improvement heuristics, other than finding a good initial solution, another big challenge for achieving good results in solving the JSP lies in finding a method, be it local search or any other approach, that effectively manages to apply modifications upon the initial solution and improves its quality. Many different approaches to improve initial solutions have been proposed in the literature throughout the years, with a wide range of different techniques that are often combined into a large exploratory framework that attempts to create strong and diverse neighborhoods (CHAUDHRY; KHAN, 2016).

2.4.1 FJSP Literature Review

In a proposal for solving the FJSP, Brandimarte (1993) split the original problem into routing and scheduling subproblems and solves both by using Tabu Search. A set of five different dispatching rules are used to populate initial solutions that later will be improved by a Tabu Search algorithm that comprises 3 memory levels: short-term, medium-term and long-term. By using these memory levels to guide the heuristic throughout its execution, two approaches are proposed: one-way and two-way. On the one-way approach, both subproblems are solved independently. On the two-way approach, the routing and scheduling subproblems communicate to each other, with information acquired in one helping the heuristic in trying to improve the other subproblem. Results are presented but are not compared to any available result in the literature in that period.

A Branch-and-Bound formulation for the JSP can be seen in Brucker, Jurisch and Sievers (1994). The authors make use of two widely established conceptual bases regarding the JSP: the disjunctive graph formulations of Roy and Sussmann (1964) (for which an example can be seen in Figure 2.2) and the block approach of Grabowski, Nowicki and Zdrzałka (1986). The authors' enumerative scheme searches for a solution by *'fixing'* the disjunctions in the disjunctive graph, i.e turning the undirected arrows into directed arrows. Their proposed algorithm can be represented by a search tree that starts from the root. The root has no disjunctions fixed, meaning it represents all feasible solutions to the problem. The successors of the root are obtained by fixing the existing *'unfixed'* disjunctions. Each successor node will correspond to a disjunctive graph that represents all solutions to the problem that respect the *'fixed'* disjunctions in the node. Subsequently, every successor is recursively handled in the same way. The examination of a nodes' search tree will continue until every disjunctive arc in the graph is fixed or it can be shown that the node does not contain an optimal solution. The branching scheme is based on the previously mentioned block approach proposed by Grabowski, Nowicki and Zdrzałka (1986). The algorithm splits the schedule in the critical path of every solution into blocks, where each block must contain at least 2 nodes and at most the maximal number of operations to be processed on the same machine. Once these blocks are defined, permutations are performed by swapping operations between the defined blocks, generating new disjunctive graph representations of the problem that can now be *'fixed'*. Results proved that the algorithm had high quality in both speed and problem-solving capabilities, having solved a 10x10 benchmark instance that had remained unsolved in the literature for more than 20 years at the time of publication.

In highly constrained problem environments such as ours, it can be quite difficult to improve upon initial solutions by applying improvement heuristic techniques, since this high number of constraints needs to be verified before any move is made, guaranteeing that the performed move is feasible. Hence, constructive metaheuristics such as ACO are viewed in high regard for solving this type of problem. A method for solving the FJSP using an ACO metaheuristic with guidance from previous solutions obtained by the algorithm is presented in Xing et al. (2010). What the authors describe as *knowledge* is an implementation of auxiliary matrices containing information from previous solutions found that improved the current best. The probability of a job being assigned to a machine with a given priority is calculated based on this data. To display the effectiveness of the proposed method, the algorithm is benchmarked against the same algorithm without the implemented learning method. Results encountered with the learning method were, in fact, superior. An important aspect demonstrated by this paper is the fact that data mining techniques can assist when dealing with scheduling problems.

Many authors (DELL'AMICO; TRUBIAN, 1993) (NOWICKI; SMUTNICKI, 1996) (GONCALVES; MENDES; RESENDE, 2005) have acknowledged that the critical path of a solution (in this case, the schedule of the machine which finishes its assigned operations last) is the best starting point for a good local search performance on scheduling problems. Another desirable aspect for good LS performance is the quality of the chosen neighborhood function. In Mastrolilli and Gambardella (2000), two are introduced. The proposed neighborhood functions are designed based on removing an operation on the critical path of a solution and reinserting it in the best possible way. Both functions are proven to be optimum connected, meaning an optimal solution can always be reached in a finite number of steps traveled across the neighborhood of an initial solution. The use of these neighborhoods with a Tabu Search algorithm lead to the discovery of 120 improved upper bounds out of 221 benchmark problem instances evaluated.

In Li, Pan and Suganthan (2010), the authors propose a hybrid Tabu Search algorithm with what they call an efficient neighborhood structure, dividing their method into 3 steps: (i) Initialization of solutions based on 4 dispatching rules for routing and 4 for scheduling; (ii) Assembly of the neighborhood structure with 3 different adaptive approaches; (iii) and lastly, applying local search to the assembled neighborhood structure with a newly proposed method called *insert and swap*. Results obtained by the proposed method were compared against several different approaches found in the literature. The proposed algorithm was shown to be both robust and fast, being able to improve results found in the literature for all instances evaluated with significant improvements in computational cost expended.

In recent years, the combination of two or more heuristics, or even exact approaches is becoming increasingly common. A wide variety of combinations have been displaying good results and flexible algorithms, capable of adapting to variations of the initial problem with ease. A solid example can be seen in XY and Liang (2016), where a hybrid GA and TS algorithm combines the diversification offered by GAs and the intensification provided TS. A wide range of comparison with results found for open instances are made and state-of-the-art literature results are obtained, validating the assessment made by the authors that the proposed algorithm performs as well as state-of-the-art algorithms with better computational time in most cases, although the authors acknowledge the fact that a fair comparison in computational time is difficult to achieve. Sundar et al. (2017) make use of a different, swarm intelligence based metaheuristic, the Artificial Bee Colony (ABC), and combine it with local search to provide a new algorithm for solving the Job Shop Problem with No-wait Constraints (JSPNW), an extension of the JSP where no waiting time is allowed between operations for a given job. The well-designed proposed communication of the ABC algorithm with the local search led to superior results when compared against two state-of-the-art algorithms found in the literature on a reasonably-sized set of benchmarking instances.

2.5 Vehicle Routing Problem

The Vehicle Routing Problem's first appearance in the literature is credited to Dantzig and Ramser (1959), in the form of the Capacitated Vehicle Routing Problem (CVRP). Their problem consisted in finding an optimal routing for a homogeneous fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. Over the years the CVRP has become the most studied version of VRPs.

Since then, a vast number of similar problems have been proposed by increasing the number of constraints in the original problem (GOLDEN; RAGHAVAN; WASIL, 2008):

- Period Vehicle Routing Problem (PVRP): Extension of the classic VRP where routes must be designed for a planning horizon consisting of 1 or more periods.
- Vehicle Routing Problem with Time Windows (VRPTW): Customers may only be served within a specified time interval.
- Heterogeneous Fleet Vehicle Routing Problem (HVRP): Routes must be designed respecting the capacity of each vehicle. If a vehicle cannot reach a set of clients for a given reason, the problem turns into the Site-Dependent Vehicle Routing Problem (SDVRP)
- Vehicle Routing Problem with Split Deliveries (SDVRP): Customers' demand may not be serviceable by a single vehicle. Hence, as in the MWSRPDT, a customer may be visited more than once.
- Open Vehicle Routing Problem (OVRP): Same as the CVRP with the relaxation that vehicles do not have to return to the depot at the end of service.
- Multi-depot Vehicle Routing Problem (MDVRP): Each customer may be serviced by a vehicle originating at any of the available depots.

In recent years a lot of focus has been given to the so-called *rich* VRPs i.e problems that are modeled after real-life problems. Often these real-life added constraints appear in the form of time and distance factors, use of heterogeneous fleets, linkage with inventory and scheduling problems, etc (CRUZ et al., 2014). For that reason, even though VRPs have been thoroughly studied in the last five decades, studies on some of its variations are still in their infancy.

As in the JSP, a wide range of objective functions for VRPs can be defined, with minimizing the maximum time taken to service all costumers, minimizing the number of vehicles/routes needed to service all costumers or minimizing variation in travel time and vehicle load being good examples of commonly used objective functions.

Due to VRPs being an extension of the TSP, naturally, common approaches to solve them were derived from works done on finding solutions for the TSP (TOTH; VIGO, 2014). Exact methods for VRPs have evolved from simpler tree search methods (CHRISTOFIDES; EILON, 1969) based on Branch-and-Bound to column generation (DESROCHERS; DESROSIERS; SOLOMON, 1992) and eventually to Branch-and-Cut-and-Price algorithms at the turn of the century (KOHL et al., 1999) (FUKASAWA et al., 2006) (PESSOA; ARAGAO; UCHOA, 2008) that continue to be improved (PECIN et al., 2017). As far as heuristics approaches go, ever since its first appearance in the literature, VRPs have had an immense number of different heuristic solution proposals in the literature, using a wide variety of constructive and improvement heuristics. In more recent decades, with the rise of metaheuristics, researchers have been able to present algorithms capable of reaching results with less than 1.0% distance from best-known values of problem instances. However, only relatively small instances can be solved to such standards. For instances with more than 300 customers, there is still a lot of room for improvement.

Metaheuristics used for solving VRPs hardly differ from the ones used on the JSP. Tabu Search (GENDREAU; HERTZ; LAPORTE, 1994) (CORDEAU; LAPORTE; MERCIER, 2001) (ESCOBAR; LINFATI; BALDOQUIN, 2014), Simulated Annealing (CHIANG; RUSSEL, 1996) (WEI et al., 2018), Ant-Colony Optimization (BELL; MCMULLEN, 2004) (REED; YIANNAKOU; EVERING, 2014) and Genetic Algorithms (PRINS, 2004) have all been used to great extent and have achieved significant results in near-optimally solving small to medium-sized instances of many of the VRP variations. State-of-the-art heuristic algorithm approaches to the VRP typically involve a combination of metaheuristic use with efficiently built neighborhood structures that allow vast exploration of the solution space.

2.5.1 VRP Literature Review

Dealing with the classic CVRP, Fukasawa et al. (2006) proposes a robust Branch-and-Cut-and-Price algorithm by merging two commonly used ideas in similar algorithms. Branch-and-bound approaches found in the literature would usually spend great amounts of time trying to apply cuts that often became more and more costly. By combining column and cut generation in an attempt to improve lower bounds, the authors present the robust Branch-and-Cut-and-Price algorithm. Although an exact method is proposed, the authors make use of heuristics to speed up the construction of routes. As a result, an exact algorithm that can be easily adapted to other variants of the CVRP by simply including any new constraints in the pricing routine, the proposed algorithm was able to solve 3 CVRP instances that were unsolvable to optimality until that point and obtained excellent results against state-of-the-art literature of the time.

The above work was extended in Pessoa, Aragao and Uchoa (2008) with the introduction of capacity-indexed variables in the model to better constraint in-degree and out-degree of vertices and avoid cycles and routes with total demand higher than the maximum available demand and using these variables to design a newly rich family of cuts. Extensive experiments were made and showed that the algorithm produces great results for small instances but the same cannot be said for large critical instances (instances with 1000 clients and 3 vehicles, for instance). In those cases, the algorithm was halted at 10.000 seconds after failing to deliver an optimal or near-optimal solution. However, the authors still strongly believed that the introduced family of cuts and/or similar ones still have a lot of room for improvement and could present significant results in the future.

The previous statement would be proved true by Pecin et al. (2017) and Pessoa, Sadykov and Uchoa (2018). The first introduces limited memory subset row cuts (lm-SRCs), a cutting method that has a significantly lower impact on the pricing routine while achieving similar results and manages to deliver an algorithm that solved CVRP instances with up to 360 customers that were previously unsolved by exact methods. The second tackles the Heterogeneous Fleet Vehicle Routing Problem (HFVRP) and also introduces a new type of cutting method, named Extended Capacity Cuts (ECC). The ECC is an adaptation of the Rounded Capacity Cut (RCC) introduced by Laporte and Nobert (1983) for the CVRP in which the right side of the inequality is changed due to the vehicles having different capacities in the HFVRP. The combination of the ECC with an improved implementation of the so-called Rank-1 Cuts that lowers the impact of cuts on the pricing routine amounts to an algorithm that managed to optimally solve instances with up to 200 customers and also a few larger ones. Both of these works heavily utilize dynamic programming concepts to reduce the cost of important operations performed by the algorithms and, based on the results found, it is reasonable to classify its use as a winning strategy to be considered for future research.

Unified proposals, claiming to solve more than one variety of VRPs are also found in the literature. A good example is the ALNS framework (PISINGER; ROPKE, 2007), an extension of the Large Neighborhood Search framework. The proposed framework claims to solve 5 variants of VRPs, who are also the most common ones found in the literature: CVPR, VRPTW, MDVRP, SDVRP, and OVRP. All 5 are mapped into RPDPTW (Rich Pickup and

Delivery Problem with Time Windows) and then solved by the algorithm. The ALNS is based on the *ruin and recreate* principle, wherein every iteration, a solution is *ruined* and *repaired* in a different way, and uses the Simulated Annealing metaheuristic. A wide exploration of the neighborhood is an advantage of large neighborhood algorithms, allowing exploration to run in a more structured fashion and avoiding being stuck in local minima. Seven different heuristics were used in the *ruining* and six on the *recreating* of a solution, which leads to perhaps the best acknowledgment made in the presented work: the authors observe that seldom using the best heuristic will lead to the best final result. A relaxation of allowing the algorithm to use heuristics that initially lead to *worse* results at given points was proven to offer better results in the end. Result-wise, their proposed algorithm is inferior to common algorithms found in the literature in both solution and computational cost aspects. Although that was expected since a generalized algorithm seldom competes with a specialized algorithm.

Another example of a unified approach can be seen in Vidal et al. (2014) where the authors define a unified Local Search procedure heavily based on optimization pre-processing techniques and combine it with the approach of Vidal et al. (2012) to obtain the Unified Hybrid Genetic Search (UHGS). The UHGS algorithm is a framework that combines four distinct concepts found in the literature into an algorithm that manages to be general-purpose without loss of quality result-wise. The combination of (i) hybridized GA and LS procedures where LS replaces the GAs mutation operator; (ii) solution representation without trip delimiters to perform crossover operations; (iii) the penalization of infeasible solutions and (iv) the use of a diversity-and-cost objective for solution evaluation to perform population management amounts to a solid hybridized algorithm. Benchmarked against 1099 instances of 29 different VRP variations the UHGS manages to consistently near-match state-of-the-art problem-tailored algorithms and even outperform some in CVRP, LDVRP (Load-Dependent Vehicle Routing Problem) and GVRP (Green Vehicle Routing Problem).

In a highly applicable environment in today's world such as warehouse logistics, a fairly recent work (SANTIS et al., 2018) proposes an ACO based algorithm to minimize travel distance in warehouse pickers in a real case scenario, claiming an achieved reduction of 19.845 meters walked per day among warehouse employees. The authors combine the ACO metaheuristic with the Floyd-Warshall (FW) algorithm, creating a hybrid algorithm they refer to as FW-ACO. The warehouse is modeled as a graph, all according to its real geographic specifications. In the modeled graph, the FW algorithm is used to find the smallest distance between each pair of vertices. Having these metrics, the ACO pheromone deposit formula is adapted to take into account the results from the FW algorithm when selecting the next move. For evaluation, results were also compared against two other literature approaches MMAX and Combined+. The proposed algorithm was proved capable of finding better results than both, also finding optimal solutions to 32 out of 36 evaluated instances. Computational time-wise, the FW-ACO is an improvement over MMAX and also Combined+, except for highly complex problems, a fact that decreases the robustness of the algorithm by a fair margin.

In recent years, the use of metaheuristics is evolving with the combination of two or more metaheuristics and/or hybridization with exact methods. Abdulkader, Gajpal and ElMekkawy (2015) combine the use of the ACO metaheuristic with three types of LS operators: 2-Opt, customer insertion, and customer swapping, resulting in a hybrid algorithm that solves the Multi-Compartment Vehicle Routing Problem (MCVRP). The authors present a comparison between results of the solutions found purely with the ACO metaheuristic and with the inclusion of each LS operator, in which the effects of the hybridization are made evident by the encounter of better solutions in the same amount of CPU time when all three operators are utilized. Another successful hybridized metaheuristic approach can be found in Gutierrez et al. (2018), where a combination of a Memetic Algorithm (MA) and the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic is implemented to solve the Vehicle Routing Problem with Stochastic Demands (VRPSD), a VRP formulation where the demand of each customer is a random variable. The GRASP metaheuristic is utilized as a mean of diversification, being used to create new populations based on the best solution found in the previous iterations population best individual. Results found are promising averaging less than 1% gap to the best-known solution in 33 out of 39 benchmarked instances.

An exact/heuristic hybrid is proposed in Alvarez and Munari (2017). The authors propose a Branch-and-Cut-and-Price algorithm in conjunction with the Iterated Local Search (ILS) and Large Neighborhood Search (LNS) metaheuristics for the Vehicle Routing Problem with Time Windows and Multiple Deliverymen (VRPTWMD), a variation upon the VRPTW in which service times at customers is dependent on the number of deliverymen assigned to the route that serves them. Their proposed algorithm is based on a set partitioning formulation that uses ILS and LNS for three purposes: generating columns for the root node of the exploration tree; providing an initial solution for the hybrid method and as improvement heuristics applied to each integer solution found by the exact algorithm. The use of these metaheuristics leads to the discovery of more good integer solutions at earlier stages of the algorithm's execution. Results show that the hybridized method proves capable of finding the same or higher quality solutions as the standalone Branch-and-Cut-and-Price algorithm in less computational time.

To solve a mixed fleet E-VRPTW (Electric Vehicle Routing Problem with Time Windows) with conventional, plug-in hybrid and electric vehicles, where vehicle autonomy is a concern, Hiermann et al. (2019) propose a hybrid Genetic Algorithm, Local Search and Large Neighborhood Search (HGA) metaheuristic that is once again hybridized with an exact set partitioning method. At every iteration, the local best solution found is stored in an archive and from that archive, a Set Partitioning (SP) problem is formulated and then solved by a commercial MIP solver. The algorithm managed to improve solutions found for 119 out of 168 instances in their benchmark set and show that the use of a mixed fleet can lower operational costs by 7% when compared to a homogeneous fleet. In a very similar approach, in an attempt to develop a generic algorithm for a large variety of rich VRPs with heterogeneous fleets, Penna et al. (2019) also makes use of a SP formulation designed to find a solution within a set of metaheuristic-obtained feasible solutions found by a ILS metaheuristic algorithm. The ILS makes use of a diverse neighborhood structure built with moves where vehicles assigned to a route are swapped for another. This diverse neighborhood structure leads to, probably, the most important contribution of their paper. Going against what is commonly accepted in the HFVRP literature, vehicle assignment decisions did not lead to significant solution improvements.

3 METHODOLOGY

The existence of precedence constraints has a significant impact on the difficulty of solving scheduling problems (LENSTRA; KAN, 1978). In a highly restricted problem such as the MWSRPDT, the use of exploratory/improvement heuristics becomes unpractical due to the low probability of finding feasible moves in case precedence is ignored and the high computational cost that comes with finding such moves if precedence is taken into account. Exact methods would also be costly, mostly due to the fact that finding solid bounds would demand a lot of computational time.

After consideration, it was reasonable to assume that constructive heuristics and hybrid exact-heuristic methods are the most suitable options available. For that reason, two different hybrid exact-heuristic algorithms were devised for our problem. Both use the Mixed-Integer Programming (MIP) model described in Section 3.1 and the Greedy Constructive Algorithm for the MWSRPDT, presented in Section 3.2.

3.1 Mixed-Integer Programming Model

In this section, we introduce to the reader a MIP model for the MWSRPDT, first presented in Pereira, Alves and Moreira (2020).

The model is devised for the problem definition presented in Section 1.1, of which we will provide the reader with a brief summary next: a given company provides a set of S services to clients through a set of K mobile teams. Each service is a digraph $s = (V_s, A_s)$ where V_s represents the set of tasks that amount to the service and A_s is a set of dependencies between tasks, meaning the existence of arc $(a,b) \in A_s$ warrants task a can only be executed after task b is completed. The problem is modeled on a complete graph $G = (V,E), |V| = n$, where V represents the set of customers to be serviced plus the depot, represented by a special vertex 1.

A planning horizon of 1 or more periods is considered for the completion of every task. Each client i requires the fulfillment of a service $s^i \in S$ and all the tasks that compose the requested service must be executed at a given point of the planning horizon. Each team possesses an amount $T \in \mathbb{R}_+$ of time available in each period and to move between any pair of vertices i and j, each team spends a predefined time $d_{ij} = d_{ji} \in \mathbb{R}_+$. To execute task a in a client i, the time needed by a team k is defined as $t^k_{ia} \in \mathbb{R}_+$.

Given the problem definition, the model makes use of an extended graph $G' = (V', E')$ where the vertex set V' is given by $V' = \{\{i,a\} : i \in V \setminus \{1\}, a \in S^i\} \cup \{1\}$, i.e each customer-task pair is represented by a vertex in G', plus the depot, represented by 1. For clarification, vertices in V' will be referenced by letters u and v. The edge set E' is considered complete.

Let H be an upper bound on the number of days needed to complete all requested services. In our implementation, a value for H will be found by the heuristic algorithm described in Section 3.2. The decision variables in the model are defined as follows:

- $x = \{x_{uv}^{kh} \in \{0,1\} : u \neq v \in V', 1 \leq k \leq K, 1 \leq h \leq H\}$. Variable x_{uv}^{kh} assumes value 1 if team k travels from u to v on day h, 0 otherwise. Assuming $u = \{i,a\}$ and $v = \{j,b\}$, $x_{uv}^{kh} = 1$ means that task b of j is executed right after the completion of task a of i by team k on day h.

- $q = \{q_{uv}^{kh} \in \mathbb{R}_+ : u \neq v \in V', 1 \leq k \leq K, 1 \leq h \leq H\}$. The value of variable q_{uv}^{kh} represents the moment in which team k arrives at v coming from u on day h.

- $y = \{y_{ia}^{kh} \in \{0,1\} : i \in V, a \in S^i, 1 \leq k \leq K, 1 \leq h \leq H\}$. Variable y_{ia}^{kh} assumes value 1 if task $a \in S^i$ is executed in customer i by team k on day h, 0 otherwise.

- $p \in \mathbb{Z}_+$: indicates the day in which the last task is completed.

In the extended graph, the time taken by a team to go from $u \in V'$ to $v \in V' \setminus \{u\}$ is given by:

$$d_{uv} = \begin{cases} d_{1j} & \text{if } u = 1, v = \{j,b\}, j \in V \setminus \{1\}, b \in S^j, \\ d_{i1} & \text{if } u = \{i,a\}, i \in V \setminus \{1\}, a \in S^i, v = 1, \\ 0 & \text{if } u = \{i,a\}, i \in V \setminus \{1\}, a \in S^i, v = \{i,b\}, b \in S^i \setminus \{a\}, \\ d_{ij} & \text{if } u = \{i,a\}, i \in V \setminus \{1\}, a \in S^i, v = \{j,b\}, j \in V \setminus \{1,i\}, b \in S^j. \end{cases}$$

The model is defined as follows. The objective function requires the minimization of total number of days needed to complete all requested services:

$$\min p \tag{3.1}$$

The constraints below assure that each requested service is executed at some day in the planning horizon:

$$\sum_{k \in K} \sum_{h=1}^{H} y_{ia}^{kh} = 1, \qquad i \in V \setminus \{1\}, a \in S^i. \tag{3.2}$$

The following constraints state that a task can only be executed if its predecessor tasks have already been executed. The moment a team arrives at customer i to perform task a is enforced to be either on a later day or at a later time on the same day as the completion time of any task b, granted that dependency arc (a,b) exists in A_{S^i}.

$$\begin{aligned}&\sum_{k\in K}\sum_{h=1}^{H}(T(h-1)y_{ia}^{kh}+\sum_{u\in V'\setminus\{i,a\}}q_{u\{i,a\}}^{kh})\geq\\&\sum_{k\in K}\sum_{h=1}^{H}((T(h-1)+t_{ib}^{k})y_{ib}^{kh}+\sum_{u\in V'\setminus\{i,b\}}q_{u\{i,b\}}^{kh}),i\in V\setminus\{1\},(a,b)\in A_{S^i}.\end{aligned}\tag{3.3}$$

The next two sets of constraints grant q variables assume correct values. The first set of constraints model the first visit in a teams' route after leaving the depot on a day h. In that case, no task had yet been performed by team k on that day. Hence, only travel time from the depot 1 to the customers location is considered. The latter set models the case when a visit to a customer is already present in the teams' route. Hence, other than any time spent moving between customers, the time team k spends performing tasks in these customers is also taken into account.

$$d_{1v}x_{1v}^{kh}\leq q_{1v}^{kh},\qquad v\in V'\setminus\{1\},k\in K,1\leq h\leq H,\tag{3.4}$$

$$\sum_{u\in V'\setminus\{v\}}(q_{uv}^{kh}+d_{vu}x_{vu}^{kh}+t_{ia}^{k}y_{ia}^{kh}\leq\sum_{u\in V'\setminus\{v\}}q_{vu}^{kh},\qquad v=\{i,a\}\in V'\setminus\{1\},k\in K,1\leq h\leq H.\tag{3.5}$$

To make sure every team is only scheduled within the daily available time, the following constraints are proposed:

$$q_{uv}^{kh}\leq Tx_{uv}^{kh},\qquad u\neq v\in V',k\in K,1\leq h\leq H.\tag{3.6}$$

Whenever $y_{ia}^{kh}=1$, the corresponding team should visit vertex $\{i,a\}$, which is granted by the following constraints:

$$\sum_{u\in V'\setminus\{1\}}x_{uv}^{kh}=y_{ia}^{kh},\qquad v=\{i,a\}\in V'\setminus\{1\},k\in K,1\leq h\leq H,\tag{3.7}$$

$$\sum_{u \in V' \setminus \{1\}} x_{vu}^{kh} = y_{ia}^{kh}, \qquad v = \{i,a\} \in V' \setminus \{1\}, k \in K, 1 \leq h \leq H. \tag{3.8}$$

No team can return to the depot and leave again on the same period, granted by:

$$\sum_{v \in V' \setminus \{1\}} x_{1v}^{kh} \leq 1, \qquad k \in K, 1 \leq h \leq H. \tag{3.9}$$

The day in which the last service is completed is obtained by the following constraint:

$$p \geq \sum_{i \in V' \setminus \{1\}} h x_{1i}^{kh}, \qquad k \in K, 1 \leq h \leq H. \tag{3.10}$$

As evidenced by the experiments conducted in Pereira, Alves and Moreira (2020), the models' use is only advised when dealing with small ($n \leq 20$) instances. For that reason, especially when dealing with medium to large instances, heuristic algorithms can be necessary. In light of this fact, two hybrid heuristic algorithms are presented in the following sections. Both proposed heuristic algorithms make use of the model in some fashion. The first uses a simplified version of the model, where flow conservation constraints are disregarded. While the second uses the model as an improvement mechanism applied to an existing solution.

3.2 Greedy Constructive Algorithm

To provide a valid upper bound for both our proposed algorithms, the Greedy Constructive Algorithm (GCA) for the MWSRPDT was devised and is detailed in this section.

The existence of dependencies between tasks is the key aspect to be considered when designing an algorithm for task scheduling. This occurs because a given task is only *free* if its predecessor tasks are finished. In a scenario where there is more than one team available, this aspect forces the routing subproblem to be performed in parallel for all teams. Every team needs to be *aware* of other teams' current time position in its schedule to know if a scheduled task, that may depend on the completion of another task that is to be performed by another team, is now free for visitation.

This aspect is incorporated into the GCA by performing both the scheduling and the routing for all teams simultaneously. In the GCA, initially, teams are inserted into a min-heap structure whose values represent each team's current time position in the current day. At every iteration, the team k that has the most unused time in that day is popped from the heap and given a new task *nt* from the current pool of available tasks.

Task dispatch is performed in greedy fashion. Let us define $R = \{\{i,a\} : i \in V \setminus \{1\}, a \in S^i\}$ as the set of every requested client-task pairs. Also, let $C = \{\{i,a\} \in R : \{i,a\}$ has been successfully planned for a team on a given day$\}$ be the set of completed tasks and $\sigma = \{\{i,a\} \in R \setminus C$ $: |A_{s^i}| = 0 \ \ or \ \ \{i,b\} \in C, (a,b) \in A_s{}^i\}$ be the set of available tasks at any given point in the algorithms execution. Starting with the current day set to 1, at every iteration, a new task is dispatched to a team. Let $\{j,\hat{a}\} \in C \cup \{1\}$ be the teams current vertex. The dispatched task is chosen according to the following equation:

$$\min_{\{i,a\} \in \sigma} d_{ji} + t^k_{ia} + d_{i1} \tag{3.11}$$

If the selected task $\{i,a\}$ can be performed within the daily available time T, it is added to C and the team is returned to the heap with its updated time position as the moment task $\{i,a\}$ is finished. Otherwise, the team is not returned to the heap, meaning its route is finished for the current day. If $\{i,a\}$ was successfully scheduled, another task $\{i,c\}$ that depended on $\{i,a\}$ might have become available. For that reason, after every successful scheduling, a check for available tasks is performed. If indeed one or more tasks have become available, they are included in σ.

Every new available task can only be started after its latest finished dependency is completed. To control the scheduled times for every task, a map structure *schedule* is introduced. The *schedule* map is initialized with every $\{i,a\} \in R$ pair as keys and $(day, min\_start, start, end)$ as values, where: *day* marks the day in which the tasks are scheduled, *min_start* represents the moment in which the task becomes available, and *start* and *end* are set to the moments the task starts and ends, respectively. Every time a task is successfully scheduled for a team, *schedule* is updated. The same goes for every time a new task $\{i,a\} \in R$ is available, where *min_start* for this task will be set as the highest *end* time across its dependencies $\{i,b\}, (a,b) \in A_{s^i}$.

If the heap becomes empty at any given time, meaning no team was able to perform the tasks still in σ, the current day is set as finished and the algorithm starts a new day. This process is repeated until every task in R is successfully scheduled for a team, yielding a complete solution.

An outline for the GCA is presented in Algorithm 1.

Algorithm 1: Greedy Constructive Algorithm (GCA) for the MWSRPDT

```
Data: Requested ($client, task$) pairs $R$, available teams $K$, daily available time $T$
Result: Feasible schedule for the completion of every requested task
$h \leftarrow 1$ // $h$ tracks the current day
$Sol \leftarrow \{\}$ // $Sol$ is an empty solution
Let $C \leftarrow \emptyset$ be an initially empty set that tracks completed tasks
Let $routes[k] \leftarrow ((1, null, 0, \infty))$ be a set that tracks the routes built for every team
  $k \in K$. Values in the tuple represent the client the team is currently located, the
  task being performed, the moment they arrived at that client and the moment they
  left the client, respectively. Initial client is always set as the depot 1, with arrival
  time 0 representing the start of the workday.
Let $\sigma$ be the set of available tasks, composed by the set of tasks $\{i,a\} \in R$ that
  either: (i) have no dependencies or (ii) every task $\{i,b\}, (a,b) \in {A_s}^i$ has been
  fulfilled, and $\{i,a\} \notin C$ at any given point in the algorithms execution.
Let $schedule$ be a set in the form $(h, min\_st, st, end)$ for every $\{i,a\} \in R$ where $h$
  represents the day task $\{i,a\}$ is performed, $min\_st$ is the minimum time the task
  can be started, $st$ is the moment the task is scheduled and $end$ the moment it ends.
  Values for $min\_st$ are set to 0 for every task $\{i,a\}$ that has no dependencies, $\infty$ for
  tasks that still have unfinished dependencies and
  $\max\{schedule[\{i,b\}][3], (a,b) \in A_{s^i}\}$ otherwise.
Let $t\_pos$ be a min-heap that tracks the current position $l$ in the workday for every
  team $k \in K$. Teams current time position within the current day is always set as the
  moment the team ended its last scheduled task. Initial value is set to 0.
while $|C| < |R|$ do
    if $|t\_pos| = 0$ then
        $Sol[h] \leftarrow routes$ // append built routes to the current solution
        $h \leftarrow h + 1$ // increment current day
        Reset $routes$ and $t\_pos$ to their initial values, set in lines 4 and 7,
          respectively.
    end
    Remove the team $\hat{k}$ with the smallest current position from $t\_pos$.
    Select next scheduled task $\{i,a\} \in \sigma$ for team $\hat{k}$ according to Equation (3.11).
    if If $\{i,a\}$ can be added to team $routes[\hat{k}]$ without violating $T$ then
        Insert $\{i,a\}$ to $C$.
        Remove $\{i,a\}$ from $\sigma$.
        Insert $(i, st, end)$ to $routes[\hat{k}]$, where $st$ represents the moment team $\hat{k}$ arrives
          at client $i$ to perform task $a$ and $end$ the moment it is finished.
        Update $\sigma$ with the insertion of any task $\{i,c\}$ that might have become
          available after the completion of $\{i,a\}$.
        Update values in $schedules$ for $\{i,a\}$ and $\{i,c\}$, if necessary.
        Reinsert team $\hat{k}$ with its updated position to $t\_pos$.
    end
end
return $Sol$
```

Following results found by greedy algorithms for similar problems, the GCA is able to find feasible solutions with extremely low computational effort. However, these solutions are often of very low quality.

3.3 A Two-Level Matheuristic Algorithm for the MWSRPDT

In this section, we make use of a simplified version of the MIP model (3.1)-(3.10), presented next, and devise a Two-Level Matheuristic Algorithm (TLMA) for the MWSRPDT. In the TLMA, initially teams are assigned to tasks and the day in which a team performs a task is scheduled (first-level). Following, feasible routes for the completion of every task based on the first-level assignments are planned (second-level). In case a feasible route cannot be planned for a team on a given day, information exchange between levels will occur causing a reassignment on the first level that will be repeated until feasible routes for this assignment can be planned on the second level.

3.3.1 Simplified Model

In this section we present the simplified model used on the first-level of this algorithm. This model is a relaxed version of the model (3.1)-(3.10) that does not consider the travel time spent by a team to move between clients when scheduling tasks and is defined as follows:

$$\min \; p \tag{3.12}$$

Let R remain the set of unscheduled tasks $R = \{\{i,a\} : i \in V \setminus \{1\}, a \in S^i\}$ and H be an upper bound on the number of periods required for the completion of every requested task. The following constraint requires every task to be performed at a given period in time.

$$\sum_{k=1}^{K} \sum_{h=1}^{H} y_{ia}^{kh} = 1, \quad \{i,a\} \in R, \tag{3.13}$$

The following constraint guarantees that precedence between tasks is respected. This is a relaxed form of precedence constraint, which considers only the days but not the moments within the days.

$$\sum_{h=1}^{H}\sum_{k=1}^{K} y_{ia}^{kh} \geq \sum_{k=1}^{K}\sum_{h=1}^{H} y_{ib}^{kh}, \quad i \in V \setminus \{1\}, (a,b) \in A_{S^i}, \{i,a\} \in R, \{i,b\} \in R, \tag{3.14}$$

This model underestimates time spent to serve every task scheduled for a day. Consequently, distance between customers is not taken into account. Granted, the following constraint is proposed to limit the scheduling of tasks in a given day according to the daily available time:

$$\sum_{\{i,a\}\in R} t_{ia}^{k} y_{ia}^{kh} \leq T, \quad 1 \leq k \leq K, 1 \leq h \leq H, \tag{3.15}$$

Finally, the day of completion of the last task is given by:

$$p \geq h y_{ia}^{kh}, \quad \{i,a\} \in R, 1 \leq k \leq K, 1 \leq h \leq H. \tag{3.16}$$

This simplified approach provides optimal assignments in a very fast manner. However, the routing aspect of the problem is completely ignored. Consequently, there is no guarantee that the optimal assignment provided by this model can be routed within the daily available time limit.

3.3.2 Initial Algorithm

Let Sol represent an empty solution and h' track the current day in Sol. Also, let $R = \{\{i,a\} : i \in V \setminus \{1\}, a \in S^i\}$ *initially* be the set of every customer-task pair yet to be scheduled. We emphasize on initially because R might have only a subset of its values throughout the algorithm's course. The TLMA makes use of an initial solution, granted by the GCA, to initially provide the model with a valid upper bound for the completion of every requested task. Let GCA_H be an initial upper bound provided by the GCA.

At every iteration, the model (3.12)-(3.16) (first-level) is executed considering the tasks in R, resulting in an optimal solution $\hat{y}$ for the y variables. Note that only the tasks in R are considered in the model. This is important because the model will possibly be executed many times with different values in R. Assuming that the solution returned by the model has H days, let $J^h = \{(i,a,k) : \hat{y}_{ia}^{kh} = 1, \{i,a\} \in R, 1 \leq k \leq K\}$ be the set of tuples in the form (customer, task, team) scheduled to be executed on day h, for $1 \leq h \leq H$. Note that days $1,2,\ldots,H$ in the tentative assignment provided by the first level actually correspond to days $h', h'+1, \ldots, h'+$

$H-1$ in the final solution. The algorithms' second level will try planning feasible routes for every scheduling $J^h, 1 \leq h \leq H$, provided by the first level. Please note that $H \leq GCA\_H$ will always hold, since an optimal solution for the scheduling-only part of the problem will never be *worse* than a heuristic solution that accounts for the routing aspect.

Starting on the first day, if the routing was successful, the routing for that day is added to the solution Sol at index h', every task in J^h is removed from R and the second level will move on to the next day's schedule. If the second level does not prove able to plan feasible routes for the assignments in J^h for a given day h, the following inequality is introduced in the model:

$$\sum_{(i,a,k)\in J^h} y_{ia}^{kh} \leq |J^h| - 1 \tag{3.17}$$

This inequality will act as a limit constraint for the first level, making sure that the number of scheduled tasks in day h is at most one task less than the previous assignment (where feasible routes could not be planned). Once the inequality is introduced the model on the first level is executed once again, yielding a new optimal solution $\hat{y}$. Following, the second level will try performing route planning for the new scheduling in $\hat{y}$.

This process is repeated until every task is scheduled for execution in a day h and every day's routes are planned, following that Sol is now a complete solution.

An algorithm for the TLMA is presented in Algorithm 2.

Algorithm 2: Initial algorithm for the TLMA.

```
Data: Initial upper bound $GCA\_H$
Result: Feasible schedule for the completion of every requested task
$h' \leftarrow 1$ // $h'$ tracks the current day in Sol
$Sol \leftarrow \{\}$ // $Sol$ is an empty solution
$R \leftarrow \{\{i,a\} : i \in V \setminus \{1\}, a \in S^i\}$ // Initially R gets set of every
    requested task
while $R$ is not empty do
    Execute the model (3.12)-(3.16) with the tasks in $R$
    $\hat{y} \leftarrow$ result for variables $y$ returned by the model's execution.
    Assume $\hat{y}$ has $H$ days and let $J^h = \{(i,a,k) : \hat{y}_{ia}^{kh} = 1, i \in V \setminus \{1\}, a \in S^i, 1 \leq k \leq K\}$ be the
     set of tuples in the form (customer, task, team) scheduled to be executed on day $h$, for $1 \leq h \leq H$
    foreach $\hat{h} \in 1,...,H$ do
        Try planning feasible routes for the schedule in $J^{\hat{h}}$
        if Route planning is successful then
            $Sol^{h'} \leftarrow$ planning for $J^{\hat{h}}$
            $R \leftarrow R - \{\{i,a\} : (i,a,k) \in J^{\hat{h}}\}$
            $h' = h' + 1$
        else
            Include constraint (3.17) in the model.
        end
    end
end
return $Sol$
```

3.3.3 Subsequent Modifications

After experimenting with this initial algorithm, we observed the occurrence of some undesirable situations that would often lead to subpar solutions being produced. Therefore, we improved upon this original concept and developed a slightly more complex algorithm that improves upon this initial idea. This improved version revolves around finding a large set of route plannings for each day instead of accepting the first feasible planning found. For the remainder of this section, we will discuss the set of components added to the algorithm.

Abusing the knowledge that a solution of at most *GCA_H* days exists, we divide the number of iterations the algorithm is allowed to run into equal parts for each day. Let max_iter be an integer parameter that limits the number of iterations the algorithm is allowed. For each day $1,\dots,GCA\_H$, the algorithm is given $iter\_per\_day = \frac{max\_iter}{GCA\_H}$ iterations to try and find feasible routes.

At every iteration, the model (first-level) will return an optimal schedule J for the tasks in R. This optimal schedule, however, is strongly tied to the quality of the present upper bound. In an attempt to bypass an occasionally weak initial upper bound, we introduce an artificial upper bound in the form of the $\hat{H}$ variable. This artificial value, initially set to the same value as GCA_H, will serve as the upper bound provided to the model throughout the course of the algorithm and will be adjusted when necessary. Quite often, 1 or more of the latter days of the returned schedule are left empty. When so, we artificially decrement $\hat{H}$ by 1 unit. This forces the algorithm into attempting a feasible schedule with the least amount of days possible. The excessive decrease in the value of $\hat{H}$, however, can lead to the model becoming infeasible. Whenever that is the case, $\hat{H}$ is increased by 1 unit. Another often seen scenario that can be detrimental occurs when the schedule returned by the model has significantly fewer tasks scheduled for the first day than for the following days, when applicable. When dealing with the scheduling aspect only, this situation is fine. However, when the routing aspect is included, we discovered that allowing the model to deliver schedules with too few tasks on the first day and significantly more on the following days would lead to the discovery of final solutions of very poor quality. To circumvent this scenario, at every iteration we check the schedule's characteristics. If the second day of the schedule has 50 or more percent scheduled tasks than the first day, our objective function is replaced with the following objective function:

$$\min p - \frac{\sum_{k=1}^{K} \sum_{(i,a)\in R} y_{ia}^{k1}}{|R|}. \tag{3.18}$$

This objective function will provide incentive to the scheduling of tasks on the first day of the planning horizon. Starting on the next iteration, this objective function replaces our original one and is used until either the model becomes infeasible or the current day is finished. After this, the first-level is done for the current iteration and the algorithm moves on to the routing of the provided schedule.

Starting on the first day $(h = 1)$, the route planning procedure, described in Section 3.3.4, is used to try and plan feasible routes for each team $k \in K$ according to the scheduling in J^h. Once the second-level returns the complete routes for the day h, two possible outcomes can arise: routes are either feasible for every team or infeasible for at least one. Please note that feasibility in this scenario means simply respecting the daily available time T, since precedence constraints are still respected on the routing procedure. Hence, no precedence-violating routes are returned by the route planning procedure.

If every teams' routes are completed in less than the daily available time T, they are feasible and stored in a feasible route plannings pool $\mathscr{F}$. Otherwise, if at least 1 out of K teams' routes failed to be completed in less than T, the whole set of routes is considered infeasible and stored in a different pool $\mathscr{I}$, containing only infeasible sets of route plannings. The reason the whole planning for that day is discarded in case of even a single route being infeasible is the existence of precedence constraints. Since the route building procedure handles the construction of the teams' routes in parallel, every route is heavily dependent on the other routes. Thus, storing partial solutions for a given day and attempting to combine those with other partial solutions was not something we considered promising. The possibility of finding a match would be small and the procedure would be costly.

The algorithm's following behavior is also dependent on the second-level's outcome. If feasible routes could not be planned for the assignments in J^h for a given day h, as originally planned, inequality (3.17) is added to the model. Once the inequality is introduced, the current iteration ends and the model on the first level is executed once again on the next iteration, yielding a new optimal solution $\hat{y}$ for the same value of R. Otherwise, if planned routes on the current iteration are feasible, different behavior is required. To guide the algorithm into a wider exploration of feasible plannings, a diversification-forcing constraint is added to the model. Let $\hat{k} \in K$ be a pseudo-randomly selected team and $(i, a, \hat{k})$ and $(j, b, \hat{k})$ be two also pseudo-randomly chosen tuples in J^h, where h tracks day currently being planned. Please note that both tuples belong to $\hat{k}$. The constraint is added as follows:

$$y_{ia}^{k\hat{h}} + y_{jb}^{k\hat{h}} \leq 1 \tag{3.19}$$

This constraint will enforce a change in the schedule returned by the model on the following iteration by not allowing the selected tasks to be performed by the same team again. These constraints are introduced with the goal of increasing the feasible planning count. Thus, instead of accepting the first encountered feasible planning, we store as many of these as we can and then choose from the stored amount via a selection procedure. Although these added cuts force diversification, adding too many of these can have a very limiting effect on the scheduling possibilities. Thus, at every 5 iterations, these cuts are completely removed from the model.

Another added component to the algorithm revolves around constraint (3.15). The complete disregard for travel distance would often lead to a scenario where the effect of the aforementioned cuts would take too long to show their desired effect. Hence, at given checkpoints in the algorithm execution, the change in the size of both feasible and infeasible pools since the last checkpoint would be evaluated. If less than 10% of the plannings discovered since the last checkpoint are feasible, an artificial value would be introduced to this constraint. The updated constraint is defined as follows:

$$\sum_{\{i,a\} \in R} t_{ia}^{k} y_{ia}^{k\hat{h}} \leq T - art\_travel\_time \quad 1 \leq k \leq K, 1 \leq \hat{h} \leq \hat{H}, \tag{3.20}$$

where $art\_travel\_time \in \{0, T\}$ represents the artificial value used to speed up the algorithms arrival at a point where feasible route plannings are found more often than infeasible ones. At every checkpoint, we evaluate the aforementioned condition. If it holds, art_travel_time is updated by a rate of $t\_time\_update\_rate$. Otherwise, if more than 50% of the new route plannings are feasible, art_travel_time is relaxed by the same rate. Values for these percentages were discovered through experimentation. Checkpoints were set as once every $\frac{iter\_per\_day}{20}$ iterations in our implementation.

Once the algorithm has reached the predefined number of iterations allowed for the current day h', we evaluate the size of the feasible plannings pool. If the feasible planning count is at least at a predefined percentage of the number of iterations allowed for the current day, the route selection procedure is called. Otherwise, the algorithm is allowed to continue looking for feasible plannings until such percentage is reached or until the iteration count is at double the allowed iterations count. Once one of these relaxed conditions is met, a call to the route selection procedure is also triggered.

The route selection procedure is designed to select the final route planning for the current day h' that will remain in the final solution $Sol^{h'}$. After experimenting with different criteria to define the *best* planning, the one that lead to better overall results was simply choosing the one that had the most amount of tasks performed on the day across all teams. According to this criterion, the *best* routes from the feasible pool are selected. Please note that there might be situations where the feasible pool is empty by the time the tardiest stoppage criterion is met. Independent from this fact, plannings stored in the infeasible pool are also evaluated. Thus, a planning will always be selected. Initially, a selection procedure is used to find the n infeasible plannings that are closer to being feasible. This is performed by solving the problem of finding the n plannings with the minimum overall difference, in time units, to the daily available time T. Let $f(\bar{r},k)$ be a function that returns the route cost in units of time for a team $k \in K$ in a route planning $\bar{r} \in \mathscr{I}$. Overall distance to T for a planning is given by $\min_{\bar{r} \in \mathscr{I}} \sum_{k=1}^{K} abs(f(\bar{r},k) - T)$. Out of the top n plannings in the infeasible pool $\mathscr{I}$, the one with the most tasks is selected as the best planning in the $\mathscr{I}$. This planning will still, however, have at least one infeasible route. Hence, we then call a procedure that will make those feasible by simply removing tasks from the teams that overflow T until every teams' route cost is less than or equal to T. Finally, the best plannings from $\mathscr{F}$ and $\mathscr{I}$ are compared, and the one with the most tasks is returned as the final routing for the day h'. The final route planning is then added to the final solution Sol at index h' and every task in $Sol^{h'}$ is removed from R. The value of h' is incremented by 1 unit and the algorithm will move on to the next day. This process is repeated until every task is scheduled for execution in a day and every day $1,...,|Sol|$ routed, following that Sol is now a complete solution.

Even with the addition of these components, the initial solution found by the algorithm is often encountered in a relatively fast manner. This solution, however, is seldom an improvement upon the original upper bound provided by the GCA. We abuse this notion by using knowledge provided by the initial solution found by the algorithm when allowing it to look for another. We analyze the latter found solution's characteristics and identify what we call the *critical services* of the problem. Critical services are defined as the services that had at least one of its tasks performed by a team on the final day of the latest encountered solution. Let $\Gamma = \{\{i,a\} : i \in V, a \in s^i, s \in S, s$ had a task performed on the final day of the latest found solution on any client}. The following attempt at building a solution will now have the following modified objective function:

$$\min p - \frac{\sum_{k=1}^{K} \sum_{\hat{h}=1}^{\hat{H}-1} \sum_{\{i,a\}\in\Gamma} y_{ia}^{k\hat{h}}}{|\Gamma|} \tag{3.21}$$

This objective function encourages the model on the first level to prioritize the scheduling of tasks that belong to services on the critical path in the earlier days of the planning horizon.

The algorithm will attempt to look for solutions until a predefined number of solutions (2, in our implementation) have been found without improvement upon the current upper bound. Please note that the initial upper bound is the value of *GCA_H*. However, every time a new upper bound is found, this value is updated and the solution counter reset. Therefore, the algorithm is allowed to search for at least the predefined amount of new solutions every time a new upper bound is found.

An algorithm for the final version of the TLMA is presented in Algorithm 3.

Algorithm 3: Final algorithm for the TLMA.

```
Data: Initial upper bound $GCA\_H$, number of allowed iterations $max\_iter$.
Result: Feasible schedule for the completion of every requested task.
$\hat{H} \leftarrow GCA\_H$
$iter \leftarrow 1$ // Current iteration tracker
$Sol \leftarrow \{\}$ // $Sol$ is an empty solution
$h' \leftarrow 1$ // $h'$ tracks the current day in the solution
$\mathscr{F} \leftarrow \emptyset$, $\mathscr{I} \leftarrow \emptyset$.
$iter\_per\_day = max\_iter/H$
$art\_travel\_time \leftarrow 0$
$R \leftarrow \{\{i,a\} : i \in V \setminus \{1\}, a \in S^i\}$ // Initially R gets set of every
    requested task
while $|R| > 0$ do
    Execute the model (3.12)-(3.16) with the tasks in $R$ and upper bound $\hat{H}$
    $\hat{y} \leftarrow$ result for variables $y$ returned by the model's execution.
    if Model is infeasible then
        In case there are cuts 3.17 or 3.19 in the model, remove 50% of them, selected at random.
        Otherwise, increase $\hat{H}$ by 1 unit.
        Go to line 10.
    end
    Assume $\hat{y}$ has $H$ days and let $J^h = \{(i,a,k) : \hat{y}_{ia}^{kh} = 1, i \in V \setminus \{1\}, a \in S^i, 1 \le k \le K\}$ be the set
     of tuples in the form (customer, task, team) scheduled to be executed on day $h$, for $1 \le h \le H$
    If the second day of $\hat{y}$ has over 50% more scheduled tasks than the first, swap objective function to (3.20).
    If $J^H$ is empty, decrease $\hat{H}$ by 1 unit.
    Plan routes for the scheduling in $J^1$ via the route planning procedure
    if Route planning is infeasible then
        Store $J^1$ in the infeasible planning pool $\mathscr{I}$
        Include constraint (3.17) in the model.
    else
        Store $J^1$ in the feasible planning pool $\mathscr{F}$
        Include constraint (3.19) in the model, for pseudo-random selected team $k$ and tasks $\{i,a\}$ and $\{j,b\}$.
    end
    if iter is a multiple of $\lfloor iter\_per\_day/20 \rfloor$ then
        If less than 10% of the plannings designed in the previous $\lfloor iter\_per\_day/20 \rfloor$ iterations were feasible, increase $art\_travel\_time$ by 0.1.
        Else, if more than 50% of the route plannings in the last $\lfloor iter\_per\_day/20 \rfloor$ iterations were feasible, decrease $art\_travel\_time$ by 0.1.
    end
    if $iter > iter\_per\_day$ and at least $0.025 \times iter\_per\_day$ feasible route
     plannings were designed for the current day $h'$ or $iter > iter\_per\_day \times 2$
     then
        Trigger the route selection procedure for the current day $h'$ and set the selected route planning as final for day $h'$ in $Sol$.
        Remove tasks in the selected route planning from $R$.
        Remove any ocurrence of cuts (3.17), (3.19), and (3.20), from the model.
        Reset $iter$, $art\_travel\_time$, $\mathscr{I}$, and $\mathscr{F}$, to their respective starting values.
        $h' \leftarrow h' + 1$
    end
    $iter \leftarrow iter + 1$
    Remove from the model any cut 3.19 that has been in place for longer than 5 iterations.
end
```

3.3.4 Route Planning Procedure

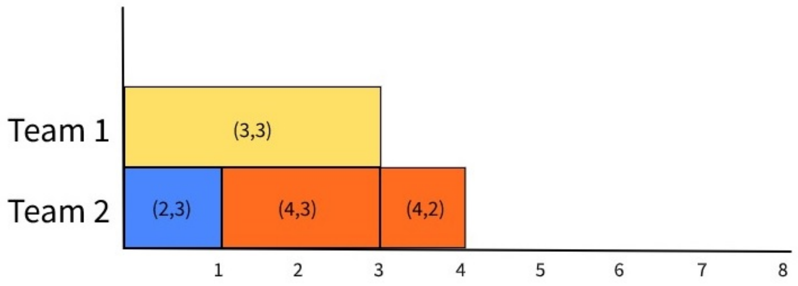
In this section, we present the route planning procedure used on the algorithm's second-level. This procedure is an adaption of the GCA, described in Algorithm 1, where the next selected task assigned for a team can only be chosen from the tasks assigned for each team k on day h by the model's execution on the first level. Only two other singularities are added. Since an optimal scheduling for teams based on task times has been delivered by the first-level, task times are omitted from the dispatching rule. As a result, only travel time between customers is considered. Recall that σ is the set of available tasks at any given point in the algorithm's execution. The dispatching rule used is as shown in the following equation:

$$\min_{\{i,a\} \in \sigma} d_{ji} \tag{3.22}$$

The only other singularity is the addition of a local search based procedure, triggered after the events of line 20 in Algorithm 1. As we have previously discussed, applying local search-based algorithms is a difficult task when dealing with problems that incorporate precedence constraints. In an attempt to circumvent this difficulty and still use the benefit provided by this established technique, we devised the following procedure.

Consider an example problem where a service provider offers a single service $s \in S$ composed of 3 tasks, for simplicity. Task 1 depends on tasks 2 and 3. Thus, digraph for the service is then defined as $s = \{\{1,2,3\},\{(2,1),(3,1)\}\}$. Since only one service is provided, every customer has requested the same service. In our example, there are four customers plus the depot, therefore $n = |V| = 5$. There are $K = 2$ available teams and daily available time $T = 8$. For simplicity reasons, let us have arbitrary task times for both teams, travel times between customers and an also arbitrary dispatching rule for the selection of tasks. Let us also define our initial value for $R = \{\{2,1\},\{2,2\},\{2,3\},\{3,1\},\{3,2\},\{3,3\},\{4,1\},\{4,2\},\{4,3\},\{5,1\},\{5,2\},\{5,3\}\}$ as the set of every requested task. Following, suppose that the model returned an optimal scheduling for the tasks in R that amounts to a single day. Let $J^1 = \{(2,1,1),(3,3,1),(4,1,1),(5,2,1),(3,2,1),(5,1,1),(4,3,2),(2,2,2),(2,3,2),(4,2,2),(3,1,2),(5,3,2)\}$ be the scheduling provided by the model's (3.12)-(3.16) execution on the first level. The second level will now attempt to plan feasible routes for this scheduling.

Figure 3.1 – Graphical representation of the route planning procedure until task $\{4,1\}$ becomes available.



Source: The authors (2020)

On the second level, due to the existing precedence constraints, the initial set of available tasks is $\sigma = \{\{2,2\},\{2,3\},\{3,2\},\{3,3\},\{4,2\},\{4,3\},\{5,2\},\{5,3\}\}$. Moving forward with our example, we arbitrarily plan the visits for the performance of these tasks. Team 1 will initially visit node $\{3,3\}$ and perform the task with a total duration of 3 time units. Team 2 will visit nodes $\{2,3\}$, $\{4,3\}$ and $\{4,2\}$, respectively, consuming a total of 4 units of time. Figure 3.1

illustrates this scenario. Upon the completion of task $\{4,2\}$ at moment 4, task $\{4,1\}$ has become available with a minimal start time set as the termination time of its latter finishing predecessor. In this case, moment 4. Up until the moment where task $\{4,1\}$ became available, every planned task was a free task. Namely, a free task at a given point in the algorithm's execution is a task that had either no predecessors or had all of its predecessors completed. This is the key aspect of this approach. The partial route planning described above is precedence-free. Since that is the case, the use of improvement heuristics is a lot easier. In our example, the partial route planning for team 2 consists of $\{\{2,3\} \rightarrow \{4,3\} \rightarrow \{4,2\}\}$. However, routes $\{\{4,2\} \rightarrow \{2,3\} \rightarrow \{4,3\}\}$, $\{\{4,2\} \rightarrow \{4,3\} \rightarrow \{2,3\}\}$ and $\{\{2,3\} \rightarrow \{4,2\} \rightarrow \{4,3\}\}$ are some examples of also valid routes for team 2. The described precedence-free scenario holds for every partial route planning performed from the moment where the last task became available until the next moment a new task becomes available. Therefore, we explore the solution space of this precedence-free task subset with the use of an improvement heuristic, which we describe next.

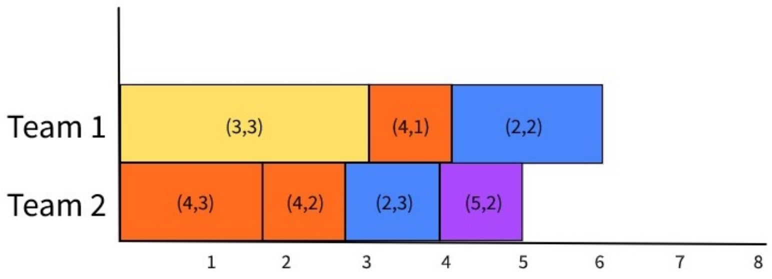
Algorithm 4: The partial route improvement heuristic.

```
Data: Partial route planning $s$.
Result: Possibly improved partial route planning $s$.
1  $improvement \leftarrow True$
2  while $improvement$ do
3      $improvement \leftarrow False$
4      $N(s) \leftarrow$ 2-opt neighborhood of $s$
5      $s^* = \{s^* : f(s^*) < f(s'), s' \in N(s)\}$
6      if $f(s^*) < f(s)$ then
7          $s \leftarrow s^*$
8          $improvement \leftarrow True$
9      end
10 end
11 return $s$
```

The improvement heuristic we employ is a local search algorithm that travels through the solution space of the 2-opt neighborhood of the partial route planning s. The improvement heuristic, described in Algorithm 4, will look for improved solutions in the 2-opt neighborhood solution space $N(s)$ until its local optimum s^* is reached. If s^* is an improvement upon the original partial route planning, s is updated. The objective function f applied to the candidate solution set evaluates the number of units of time used by the partial route. For exemplary reasons, let partial route $s^* = \{\{4,3\} \rightarrow \{4,2\} \rightarrow \{2,3\}\}$ have a fitness value less than $f(s)$. Therefore, s^* is defined as the final planning for this subset of tasks and the algorithm moves on with the planning of the remaining tasks. At the end of this procedure, current partial route plannings for each team are set as final and will remain unchanged. Following, empty partial plannings are initialized for each team and the algorithm continues its execution.

Continuing with our example, let $\{4,1\}$ be the next planned visited for team 1 and $\{5,2\}$ be the next visit made by team 2. Following, team 1 will then be dispatched to visit node $\{2,2\}$. Hence, the current partial route planning for teams 1 and 2 are $\{\{4,1\} \to \{2.2\}\}$ and $\{\{5,2\}\}$, respectively. At the moment task $\{2,2\}$ is completed, task $\{2,1\}$ becomes available. A snapshot of the route planning until this point is illustrated in Figure 3.2. Consequently, a call to the improvement heuristic is triggered on the partial route $s = \{\{4,1\} \to \{2,2\}\}$ planned for team 1. In this case, $N\{s\} = \{\{2,2\} \to \{4,1\}\}$, since only one other route is possible with this subset of tasks. This behavior is continued until the route planning is complete. Please note that the procedure is triggered only for the current partial route of the team who has been scheduled to visit the task that recently became free. In this last case, team 1, who was scheduled to serve task $\{2,1\}$ by the model's execution on the first-level. In spite of this fact, current partial routes for every team are set as final and new ones initialized. This is required due to the aspect of the route plannings being performed in parallel, in order to avoid violating precedence constraints.

Figure 3.2 – Continued graphical representation of the route planning procedure until task $\{2,1\}$ becomes available. Please note that task $\{4,1\}$ could be started earlier due to the improvement found by the procedure in the partial route for team 2.



Source: The authors (2020)

As a result, the route planning for a team k at day h is given by a sequence of precedence-free partial plannings. Combining those for every team will amount to the full route planning for day h.

3.3.5 Parameterization

The dynamic nature of this algorithm lead to a substantial need for parameter experimenting. In this section, we will discuss this experimenting and how the final parameters were defined.

The initial upper bound plays a heavy role in the quality of the final solution found by the algorithm. A weak initial upper bound would often increase the running time of the algorithm by substantial amounts, since the algorithm is designed to find at least two complete solutions without improvement as a stoppage criterion. This initial upper bound is used as an assurance that a feasible solution can be found with at least GCA_H days. However, with the removal of the routing aspect in the simplified model used in this algorithm, the optimal scheduling returned by the model would often have unused days with a planning horizon of size GCA_H. In order to tighten this bound even more and guide the algorithm towards finding an improved solution, an artificial upper bound variable $\hat{H}$ is initialized with a day removed from the initial bound. The assumption that a feasible solution could be found with at most GCA_H-1 days is fairly reasonable, due to the simplistic nature of the GCA. Consequently, solution quality showed significant improvement after tightening this initial bound. This aggressiveness in dealing with the size of the planning window the algorithm is allowed can also be too strong at times. By combining this adjustment with the added cuts that would occur at every iteration of the algorithms execution, the model would occasionally become infeasible. Thus, the value of $\hat{H}$ is tightened and relaxed many times throughout the course of the algorithms execution.

Another pair of aggressive characteristics the algorithm possesses are the added cuts (3.17) and (3.19). Both can also lead to infeasibility by their inherent characteristics of greatly diminishing the space of possible assignments. Hence, to try and lower this aggressiveness while still maintaining their desired effect, these cuts were dynamically removed at given points in the algorithm. Cuts (3.17) are less aggressive and, therefore, are only removed *partially* if the model becomes infeasible. Whenever that is the case, we remove 50% of these added cuts chosen with pseudo-random probability. As for (3.19) cuts, 100% are removed anytime the model becomes infeasible. Also, at every 10 iterations the latter are, again, fully removed. Values for these parameters were found through heavy experimenting. With these, we observed a good trade-off between the need for dynamic checks and updates against the diversification they offer. Whenever infeasibility would occur on the model, $\hat{H}$ would also be relaxed by 1 unit. However, if an attempt to increment $\hat{H}$ would lead to $\hat{H}$ having a value higher than the initial upper bound *GCA_H*, the current partial solution is discarded and the the current solution building attempt is restarted. The next solution attempt, however, will have any service executed on the final of the latest attempt prioritized as shown in (3.21).

The rate and frequency in which the artificial travel time *art_travel_time* is updated is perhaps the most sensitive parameter in our algorithm. When dealing with the frequency of the checkpoints in which an update may take place, we observed that a high number of checkpoints lead to better responsiveness of the algorithm. By checking if updates to *art_travel_time* are needed more often, we observed that the algorithm managed to reach a point where feasible plannings are more often encountered faster than with a low number of checkpoints. As for the rate in which this parameter is updated, our observations showed that relatively small updates worked well. Especially due to the high number of checkpoints we employ, lower update rates allowed the algorithm to increasingly reach the point of stabilization i.e often finding feasible plannings with reasonable use of daily available time. Higher values displayed a tendency to lead the algorithm into finding feasible, but low-quality plannings. In our implementation, the ideal value was found to be 0.1 units of time per update.

In regards to computational time spent by the algorithm, a final parameter must be discussed. As we have previously stated, the algorithm is parameterized to look for 2 solutions without improvement as a stopping criterion. If a new upper bound is found, however, this counter is reset. This setting comes with an evident increase in computational time spent. However, in most cases, the algorithm's ability to find a feasible solution is relatively fast. Thus, we decided that this parameterization carries a good trade-off between computational time spent and final solution quality in most instances.

3.4 A Sliding Window Improvement Algorithm

As we have previously stated, the high volume of dependencies in our problem makes the use of improvement heuristics such as Local Search very costly. In this algorithm, we make use of the model (3.1)-(3.10) as a Local Search operator based on an initial heuristic-constructed solution and propose a Sliding Window Improvement Algorithm (SWIA) for the MWSRPDT. By using the model considering only the customer-task pairs in a small window of days within the incumbent solution, the computational effort required for the models' execution will be significantly lower while still providing an optimal solution for that time window.

3.4.1 Formal Definition

Let Sol be an initial solution with H days and $\hat{H}$ be an integer value $1 \leq \hat{H} \leq H$ that will artificially represent the size of Sol, initially set to H. Also, let w represent the size of a time window in $\hat{H}$ where $1 \leq w \leq \hat{H}$. Note that the value of $\hat{H}$ will change throughout the course of the algorithm. The algorithm starts by considering the whole incumbent solution of size $\hat{H}$ and sets $R = \{\{i,a\} : \{i,a\}$ is executed in Sol on day h by a team $k, \hat{H} - w \leq h \leq \hat{H}\}$ as the set of customer-task pairs scheduled in the $\hat{H} - w, \ldots, \hat{H}$ days of the incumbent solution. The full model is then used on R, yielding an optimal solution $\widehat{Sol}$ for the tasks in R. If the first day of $\widehat{Sol}$ has more tasks than the first day of the time window in Sol, or the number of days in $\widehat{Sol}$ is less than w, the solution provided by the model is an improvement upon the original one in Sol and therefore will replace the part of the solution in Sol that is bounded by the time window $\hat{H} - w, \ldots, \hat{H}$. The value of $\hat{H}$ is then updated to the new size of Sol and a new iteration starts. If $\widehat{Sol}$ did not provide an improvement upon the original solution found in the time window in Sol, the value of $\hat{H}$ is then decreased by one unit, meaning the window w is also shifted backward by one unit in Sol. This process is repeated iteratively until $\hat{H} < w$. An outline for the SWIA is presented in Algorithm 5.

Algorithm 5: Algorithm for the SWIA.

```
Data: Initial solution $Sol$, $w = 1 \leq w \leq |Sol|, w \in \mathbb{Z}$
Result: Feasible schedule for the completion of every requested task
1  $\hat{H} \leftarrow |Sol|$
2  while $\hat{H} \geq w$ do
3      $R \leftarrow \{\{i,a\} \in Sol^{kh}, \hat{H} - w \leq h \leq \hat{H}\}$
4      $\widehat{Sol} \leftarrow Model(R)$ // $\widehat{Sol}$ gets the result of the model (3.1)-(3.10)
           execution given $R$
5      if The number of tasks in the first day of $\widehat{Sol}$ is larger than the number of tasks in
         the day $\hat{H} - w$ of Sol or the number of days in $\widehat{Sol}$ is smaller than $\hat{H} - w$ then
6          Replace days $\hat{H} - w, \ldots, \hat{H}$ of $Sol$ with $\widehat{Sol}$
7          $\hat{H} \leftarrow |Sol|$
8      else
9          $\hat{H} \leftarrow \hat{H} - 1$
10     end
11 end
```

A graphical example of the improvements found by the algorithm during its execution is shown in Figure 3.3. Starting from an initial solution provided by the GCA and the value of w set to 2, at the first iteration, the model is executed on the last 2 days of the incumbent solution. Promptly, the first improvement is found. The two latter days from the initial solution are condensed into a single day containing the tasks originally scheduled on both. Hence, a new upper bound is found. Therefore, the algorithm will once again execute the model on the last 2 days of the newly encountered incumbent solution. At iteration 2, another improvement is found. At iteration 3, however, no improvement could be made. As a consequence, at iteration 4 the window is moved back by a day, analyzing days 12 and 13 of the incumbent solution with 14 days total. This process is repeated until at iteration 19, where the window can not be moved back further and no improvement is found. Over the course of the algorithm's run, improvements were found at iterations $1,2,4,6$ and 9. After a full run of the algorithm with the value of w set to 2, the final solution consists of 11 days, 5 less than the initial solution found by the GCA.

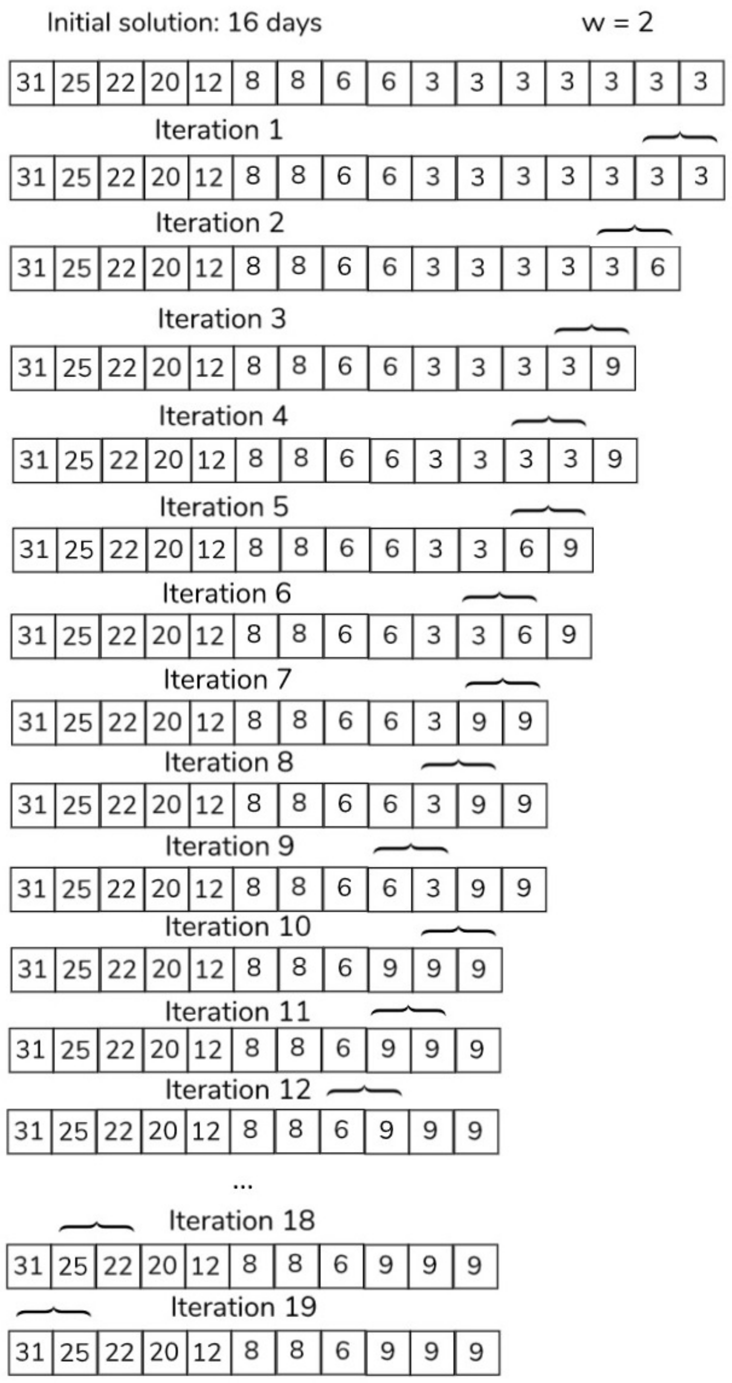
3.4.2 Parameterization

Although the SWIA makes use of the model (3.1)-(3.10) in a small window of the incumbent solution, the task of finding an optimal routing and schedule for the tasks in that window can still be quite cumbersome. For that reason, parameter selection for the MIP solver was performed by using the model on full problem instances instead of the predefined windows. The gain in computational time observed in the full problem instances followed when the model was used on a subset of tasks.

Only two parameters presented significant improvement to the solvers' performance: Branch Priority and the choice of the algorithm used for the Linear Relaxation (LR). Branch priority was set to the y variables, which lead to noticeable performance gains. However, the most gains were obtained with the choice of the algorithm for the LR in both the root node as well as on every new node encountered by the branch-and-bound procedure. The use of the Barrier algorithm instead of Simplex/Dual Simplex lead to a heavy decrease in computational time spent on solving most instances. In some, while the Simplex/Dual Simplex algorithms could not solve the root node's LR in 3600 seconds, the Barrier algorithm was able to find solutions in 5 seconds or less.

Following the implementation in Pereira, Alves and Moreira (2020), another performance gain was obtained by making use of the solvers' callback functionality. Every time a new node was found, the callback function was used to try finding a new upper bound using an adaptation of the GCA. At every new node, the adapted GCA would build a heuristic solution

Figure 3.3 – Graphical representation of a SWIA execution with $w = 2$ on instance 3 of type A with 60 customers. Squares represent days and the value inside each square counts the number of tasks scheduled and routed on that day. Brackets highlight the window being solved by the model at every iteration.



Source: The authors (2020)

for the tasks in the current window by choosing the next dispatched task according to the current values of the x and y variables in the current node. For example, say team k is currently executing task j in client i on day h, next visit is selected according to $\min_{\{\hat{i},\hat{j}\}\in\sigma} x^{kh}_{\{i,a\}\{\hat{i},\hat{j}\}} + y^{kh}_{\hat{i}\hat{j}}$, where σ is the set of available tasks at a given point in the algorithms execution. If an improved upper bound is found by the adapted GCA, it is set as a new incumbent solution.

Finding an ideal value for the size of the window w was also challenging. Let H be the initial upper bound found by the GCA. If w set to 1 and $H = 1$, the optimal solution would always be found or proven optimal in less than 5 seconds. Please note that the model was always used with a warm start, using the incumbent solution's values for the problems' decision variables. For $H = 2$, w will remain set to 1, initially. However, after each day is optimized individually, w is updated to 2. Consequently, anytime the algorithm finds a solution with $w = H$, that solution will be optimal. For $H > 2$, however, initializing $w = 1$ is quite limiting, meaning the model would be limited to improving the schedule for a single day. For that reason, an initial value of at least $\min[H, 2]$ is advised for w. This value allows the model to work with a scope that is relatively small while still being somewhat impactful when the value for H is higher. This value, however, remained insufficient in some instances. For that reason, we decided to allow the value of w to be increased by 1 unit until $w \leq \lfloor \frac{|Sol|}{2} \rfloor$. This allowed the algorithm to overcome the small window limitation of instances where the solution spans across several days. However, often the algorithm would reach its time limit, since even relatively small windows $(w \geq 4)$ were proven to be very challenging for the model in most cases.

4 COMPUTATIONAL EXPERIMENTS

In this section, we present the problem instances used to evaluate the three algorithms presented in this document. Following, we present the results obtained by our algorithms in the set of evaluated instances followed by a findings discussion.

4.1 Problem Instances

To evaluate our proposed algorithms, we make use of three types of artificially generated instances, labeled A, B, and C, proposed in Pereira, Alves and Moreira (2020). Most features are shared among the three instance types, with small changes applied to instances of type B and C to add diversity and difficulty to the problem.

The instance generator, detailed in the aforementioned work, is given the number n of vertices, the number $|S|$ of offered services, the number $|V_S|$ of tasks in each service, the number K of teams and the workers daily available time T. To generate the complete graph G, depot and customer locations are randomly placed in a $100x100$ grid with predefined travel times d_{ij} between every pair of vertices i and j. The precedence graph for each service is generated by spreading its tasks over three groups at random. Tasks placed in the first group have no dependencies. Tasks from the second group depend on every task in the first group and tasks from the third group depend on every task from the second group. The requested service for every customer is selected at random, when applicable.

In instances of type A, every team possesses the skill necessary to perform any task. Values for $|S|$ and K are both set to 3 and the number of tasks in services S_1, S_2 and S_3 are 1, 3 and 5, respectively. Instances of type B differ from type A only by the fact that some teams are allowed to not have the proficiency necessary to perform some tasks.

Type C instances are significantly harder. In these, each task can only be performed by a single team. A single three-task service is available and $K = 3$, meaning each team is responsible for performing a single task at every client.

4.2 Results

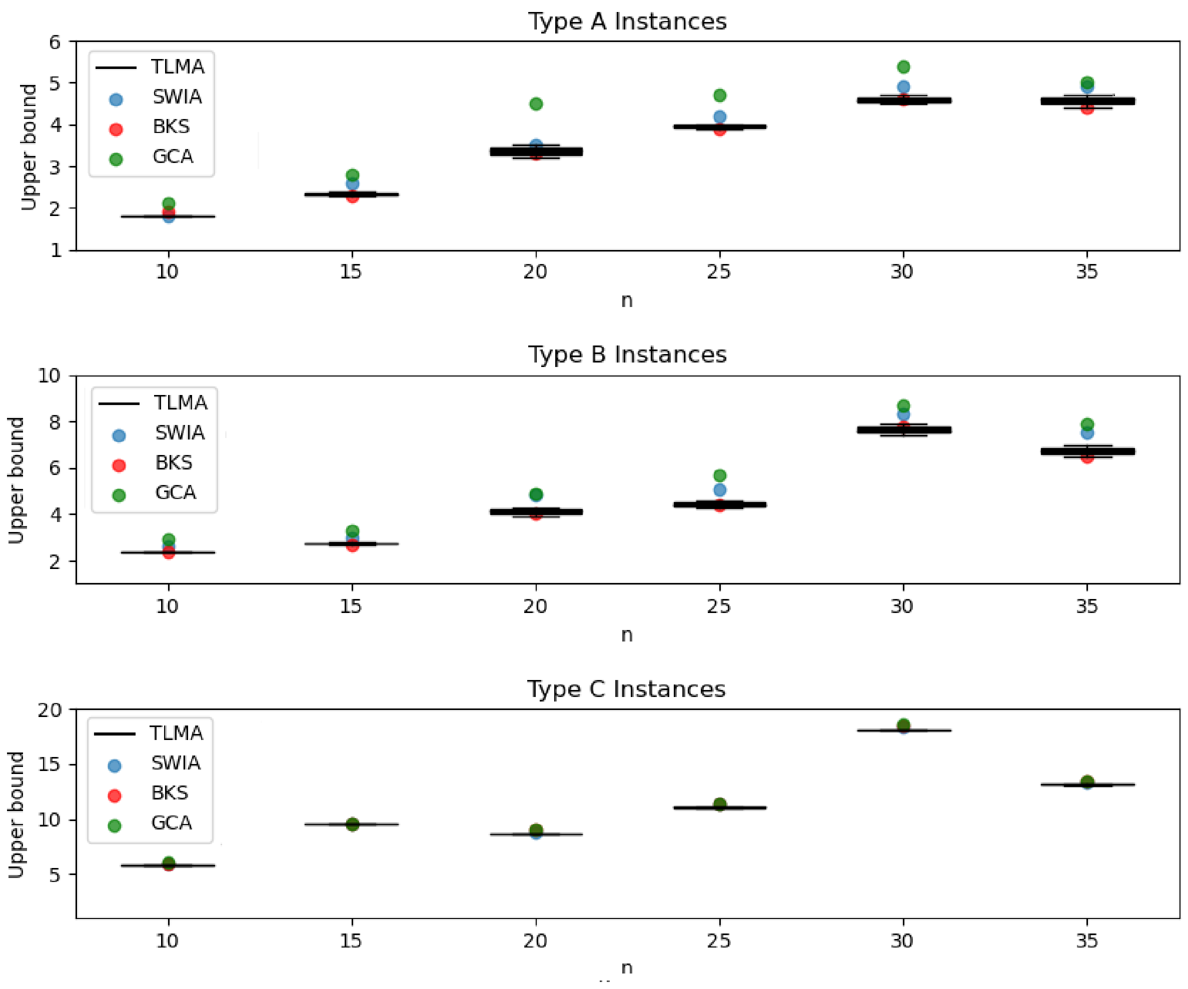
In this section, we present the results obtained by the algorithms presented in this document.

All three algorithms were implemented using the Python (ROSSUM; JR, 1995) programming language, version 3.8.2. Both models were solved using the Gurobi Optimization Package (OPTIMIZATION, 2020), version 8.0, by means of its Python API. Any parameterization regarding the number of threads used by the solver were untouched. Experiments were conducted on a machine equipped with an 8-core i7-7700 3.60GHz Intel CPU and 16GB of RAM, running on a 64-bit Ubuntu operating system, at version 18.04.3.

Following the experiments conducted where the MWSRPDT was first presented Pereira, Alves and Moreira (2020), problem instances were divided into two different sets for evaluation. The first includes small to medium-sized instances $(n \leq 35)$ and the latter contains medium to large-sized instances $(n \geq 40)$.

Results for small to medium-sized instances of type A, B and C are presented in Tables 4.1, 4.2 and 4.3, respectively. Instance information is presented in the columns under "Instance", where "n" is the number of customers, "id" is a unique identifier and "tasks" presents the number of vertices in the instance. Under "BKS" we present the Best Known Solution (in days) for that instance. Values for this column were obtained from Pereira, Alves and Moreira (2020). To our knowledge, it is the only work in the literature dealing with the MWSRPDT at the moment of writing this document. Still, individual solution values for instances of type C with $n \in \{20, 25, 30, 35\}$ are not provided in the aforementioned work. Thus, values for this column are set as "-". Results for the GCA are presented under "GCA", where the number of days in its solution lies under "ub" and the computational time, in seconds, spent to reach that solution is presented under "t". Differently from the other algorithms in this document, the TLMA carries a stochastic component, imposed by the pseudo-random choosing of activities to apply (3.19) cuts. Therefore, each instance was solved 10 times by the algorithm. Under "TLMA" we present the results found for this proposal. Columns "best" and "worst" present the number of days in the best and worst solutions found. Moreover, the average number of days across all 10 runs is presented under "avg". Computational time, number of iterations used and number of solutions found during the algorithm's execution are displayed under "t", "iter" and "sol. count", respectively. These values are also presented as averages of the 10 runs. Following, results for the SWIA are presented under "SWIA". The number of days in the final solution found by the algorithm is presented under "ub". Computational time and number of iterations are also presented under "t" and "iter", respectively. The "*" symbol under a "t" column means the maximum allowed time for the algorithm was reached, triggering an immediate stop to the algorithm. Whenever this is the case, if a solution had been found until reaching the time limit, it is presented. Otherwise, the "-" symbol is presented under "ub". Time limits were set as 1800 and 3600 seconds for TLMA and SWIA algorithms, respectively. Values displayed in bold highlight when a new upper bound (our solution has fewer days than BKS) has been discovered for an instance. Results for medium to large-sized instances are presented in Table 4.4. In such, the "id" column is omitted. Instead, values for every column are presented as averages for every value of n. Everything else follows the above description.

Figure 4.1 – Average upper bound comparison for the algorithms proposed in this document and the Best Known Solution (BKS) for the evaluated instances with $n \leq 35$. Values for the TLMA are presented in boxplot form to incorporate the spread between best, worst and average results.



Source: The authors (2020)

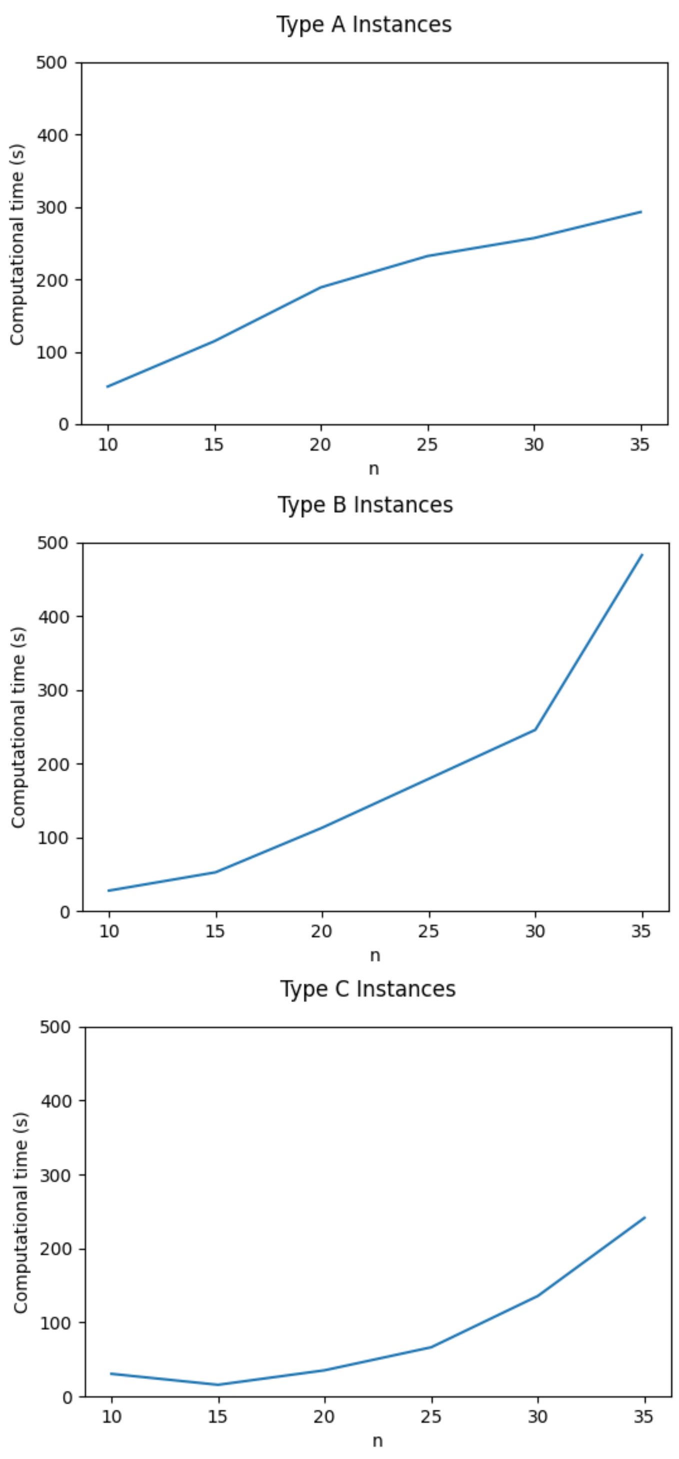
In this initial set of instances, out of the three algorithms presented in this document, the dominance of the TLMA is quite clear. Even considering only the *worst* solution found by the algorithm, its quality is still equal or superior to the ones found by either the GCA or the

SWIA. This holds for any value of n across all three instance types, see Figure 4.1. In regards to computational time, the TLMA is also the winner. Apart from instances of type A with $n \in \{20, 35\}$, computational time averages for this algorithm are far inferior to the ones for its counterpart, the SWIA. A visual representation of the growth in computational effort made by the TLMA as the value of n grows is shown in Figure 4.2. Due to its simplistic nature, we leave out the GCA in computational time comparisons.

Concerning the BKS for every instance in the first set, we can see that the SWIA falls short and does not have strong enough results to compete with the ones in the literature. Very often, even when reaching its time limit, the quality of the solution found by that point is still insufficient. Therefore, the often elevated computational time averages presented by the algorithm do not justify themselves. Still, the algorithm managed to beat the BKS average three times on type C instances. The TLMA, however, is very competitive. Across all three instance types and every value of n, the algorithm failed to match the quality of the BKS solution for an instance only twice: once for type B and once for type C. Moreover, 12 new upper bounds have been discovered: 3 for type A, 6 for type B and 3 for type C. An important thing to notice is that, although the algorithm carries a stochastic component, the difference in days from the best to the worst solution found across 10 runs is at most 1 for every instance, except instance 8 of type B with $n = 35$. In such, that difference amounts to 2 days. Although the discovery of 6 new upper bounds on type B instances is significant, we believe our strongest results for this algorithm lie in type C instances. In those, our average upper bound is of higher quality than the BKS so far for every value of n in this set, even when considering only the worst solution found for every instance.

Our choice of parameterization for the TLMA lead to a nice balance between the number of iterations spent until a complete solution is found. As shown by Figure 4.3, for all three types of instances, the average number of iterations per complete solution is very similar. As for SWIA, the number of iterations the algorithm is allowed to run is heavily dependent on the value of the initial upper bound. Our choice of allowing the size of the window w to go up to 50% of the size of the incumbent solution had a large contribution to the elevated computational times seen as the value of n rises. Allowing the size of w to reach a smaller percentage would naturally result in a lower computational effort being made. However, solution quality would also decrease.

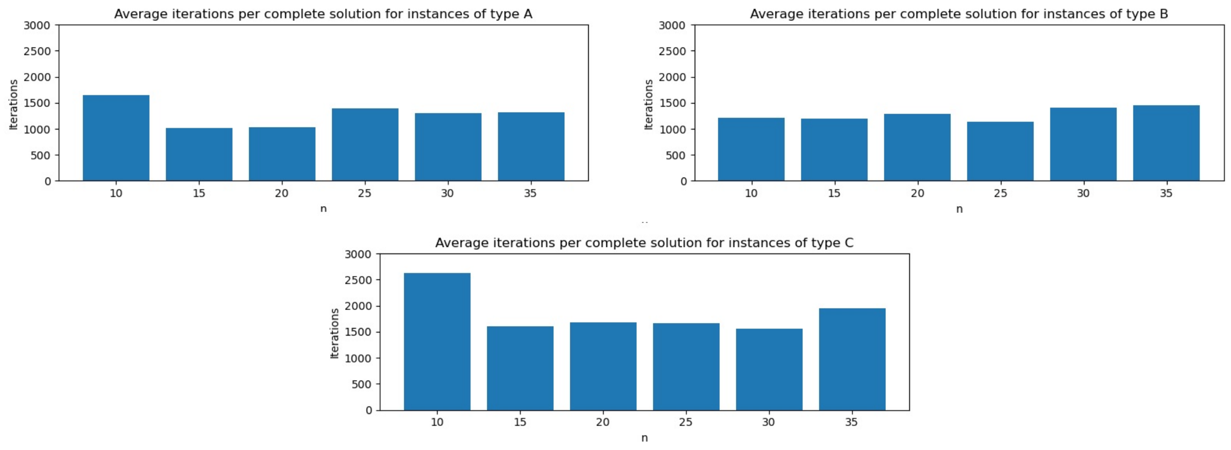
Figure 4.2 – Average computational time spent by the TLMA on small to medium-sized instances.



Source: The authors (2020)

Results on the second set of experiments strengthen our initial assumption that the TLMA is the superior algorithm among the ones presented in this document. Once again, even the worst solutions found by this algorithm are superior to the ones found by its counterparts for every value of *n* and all three types of instances. However, computational time expended

Figure 4.3 – Average number of iterations used by the TLMA per complete solution for small to medium-sized instances with $n \leq 35$. Instance 9 of types A and B with $n = 10$ were not considered for this purpose due to being relatively *easy* instances with an outlying number of solutions found.

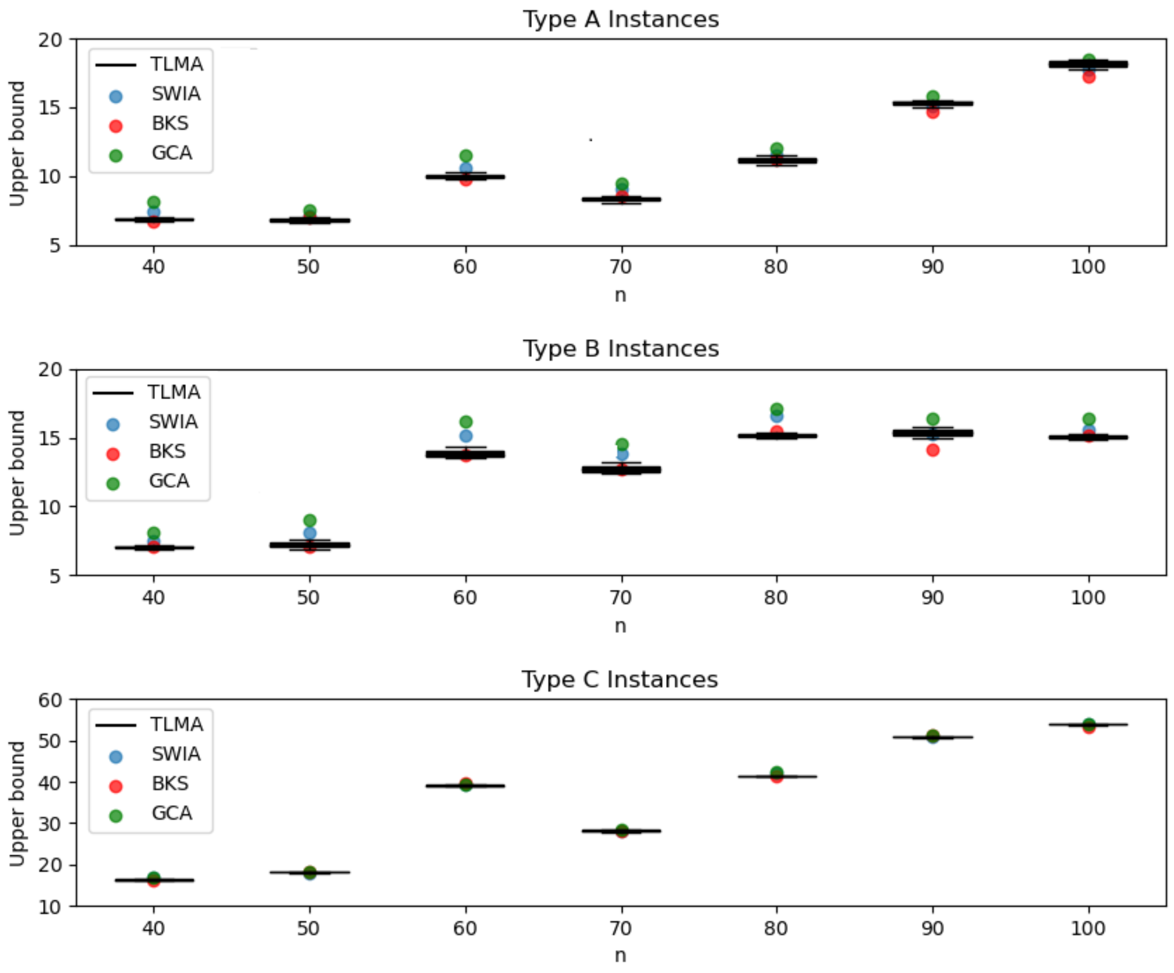


Source: The authors (2020)

by the algorithm raises a major concern. As the value of n grows past the 80 mark, the average computational time expended by the algorithm sits near its set time limit. Often, the algorithm would fail in completing a single solution within the given time limit. Whenever that is the case, the returned solution would be the one provided as an initial upper bound (i.e the same as the GCA). These often elevated computational times are due to a single reason: the use of the model in Section 3.3.1. Although it is a lightweight version of the model in 3.1 that executes with blazing fast computational times on small to medium-sized instances, as the value of n grows past this mark, those times take a severe hit. As exemplified by Figure 4.5, the number of iterations the algorithm manages to run within the time limit takes a strong downturn as the value of n grows. Consequently, the number of solutions found by the algorithm also decreases. Although the algorithm would still solve some instances with reasonable computational times and solid solutions, the use of the model can be seen as a bottleneck that does not allow the algorithm to move past this plateau.

Despite our scalability concerns, the TLMA managed to reach high-quality results on all three instance types. Most notable results lie on types B and C. In those, the algorithm failed to present an upper bound average that improves the BKS average for a value of n only once each, see Figure 4.4. Following what was observed in the first set of experiments, the SWIA performed better on instances of type C. Once again, it managed to find better solutions than the BKS for three values of n. However, once again, elevated computational times are reported.

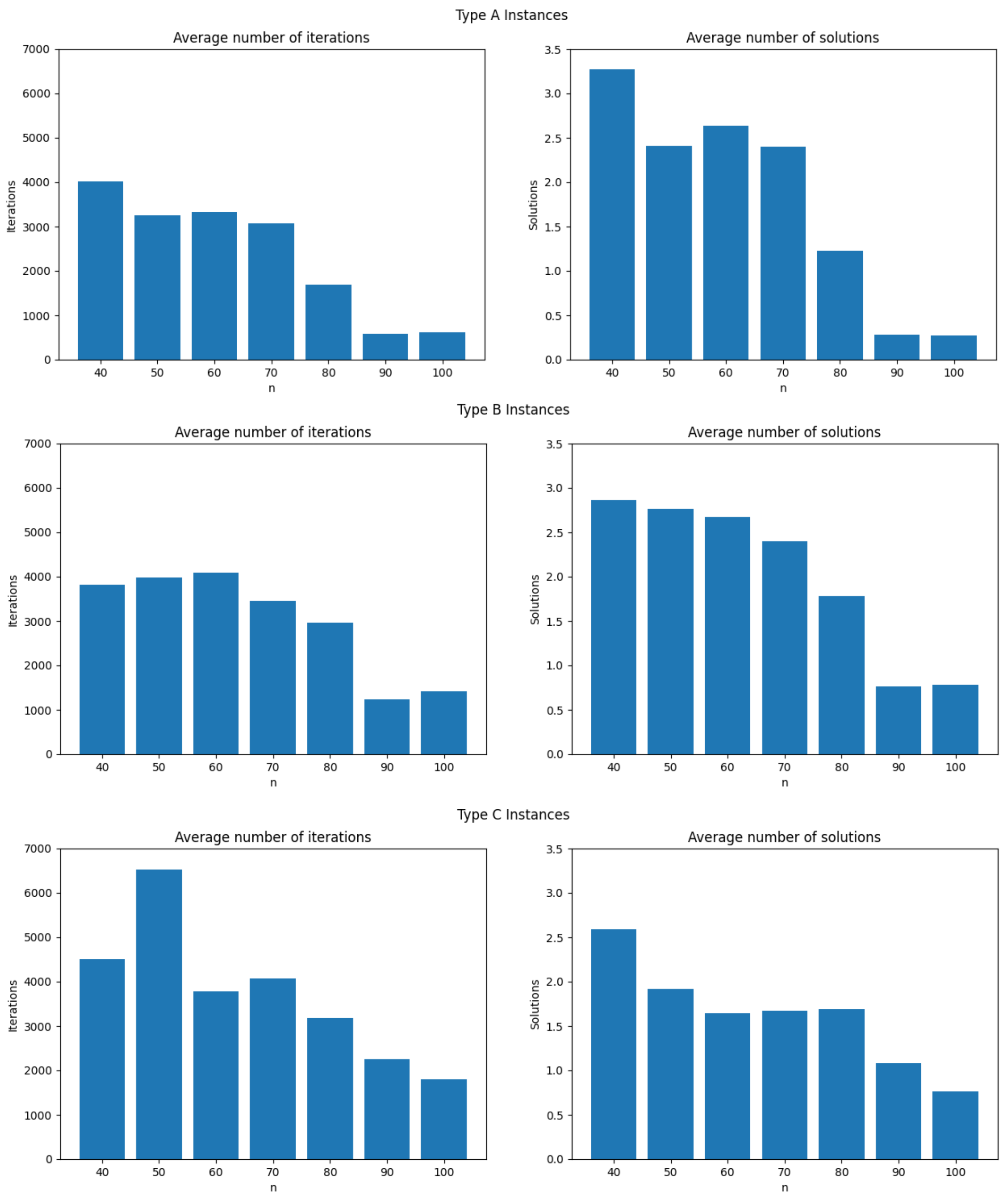
Figure 4.4 – Average upper bound comparison for the algorithms proposed in this document and the Best Known Solution (BKS) for the evaluated instances with $n \geq 40$. Values for the TLMA are presented in boxplot form to incorporate the spread between best, worst and average results.



Source: The authors (2020)

Overall, the TLMA is the more suited solution method proposed in this document for instances of all three types. It performed formidably on all three, presenting 6/13, 9/13, and 12/13 solution averages that surpass the BKS for types A, B, and C, respectively. However, as the value of n enters the 80 threshold, even the use of the simplified model becomes a nuisance. The elevated computational effort it requires makes the algorithm a less desirable method whenever low computational time is required. In that case, alternatives approaches can be more desirable.

Figure 4.5 – Average number of iterations and solutions found by the TLMA on medium to large-sized instances.



Source: The authors (2020)

Table 4.1 – Results for small to medium-sized instances of type A with $n \leq 35$

Instance				GCA		TLMA						SWIA		
n	id	tasks	BKS	ub	t(s)	best	worst	avg	t(s)	iter	sol. count	ub	t(s)	iter
10	0	23	2	2	0.00	2	2	2	75.91	6023.7	2	2	1.65	1
	1	35	2	3	0.00	2	2	2	38.16	1858.6	3	2	22.25	2
	2	31	2	2	0.00	2	2	2	75	2006.7	2	2	0.33	2
	3	35	2	3	0.00	2	2	2	45.65	1846.2	3	2	33.64	2
	4	23	2	2	0.00	**1**	**1**	**1**	8.71	1000	2	**1**	543.24	1
	5	23	2	2	0.00	2	2	2	35.19	5855	2	2	23.35	1
	6	29	2	2	0.00	2	2	2	112.55	2901.4	2	2	0.30	1
	7	29	2	2	0.00	2	2	2	79.27	6145.2	2	2	729.16	1
	8	31	2	2	0.00	2	2	2	46.24	2008.8	2	2	0.34	1
	9	19	1	1	0.00	1	1	1	1.79	1000	34	1	0.05	1
	average	27.8	1.9	2.1	0.00	**1.8**	**1.8**	**1.8**	51.84	3064.56	5.4	**1.8**	135.43	1.2
15	0	48	2	2	0.00	2	2	2	53.30	2024.9	2	2	0.84	1
	1	48	2	3	0.00	2	3	2.2	75.07	3554.5	2.8	3	*	2
	2	48	3	4	0.00	3	3	3	72.40	3175.9	3.3	4	0.61	3
	3	52	2	3	0.00	2	2	2	96.51	4567.8	4	3	538.59	2
	4	40	2	2	0.00	2	2	2	55.64	2020.9	2	2	0.57	1
	5	42	2	3	0.00	2	2	2	24.11	1661.7	3	2	7.75	2
	6	38	2	3	0.00	2	2	2	30.11	1573.9	3	2	0.79	2
	7	46	3	3	0.00	3	3	3	419.34	3589.6	2	3	0.64	2
	8	46	3	3	0.00	3	3	3	82.08	2009.2	2	3	0.63	2
	9	38	2	2	0.00	2	2	2	235.64	2349.4	2	2	0.51	1
	average	44.6	2.3	2.8	0.00	2.3	2.4	2.32	114.42	2652.78	2.61	2.6	415.09	1.8
20	0	63	5	5	0.01	5	5	5	852.44	2879	1.9	5	0.94	4
	1	63	3	5	0.01	3	4	3.8	252.55	3913.4	3.3	4	1.15	4
	2	59	4	10	0.00	4	4	4	135.46	1344	3	5	294.10	13
	3	55	3	3	0.00	**2**	3	2.7	67.40	4625.4	2.3	**2**	310.94	2
	4	51	3	3	0.00	3	3	3	81.85	2021.4	2	3	0.82	2
	5	61	3	4	0.01	3	3	3	83.68	2219.4	3.1	3	1.30	3
	6	55	3	3	0.00	3	3	3	105.91	2032.2	2	3	0.98	2
	7	63	4	5	0.01	4	4	4	128.57	3475.4	3	4	1.25	4
	8	43	2	3	0.00	2	3	2.1	42.75	2741.6	3	2	1.44	2
	9	61	3	4	0.01	3	3	3	137.59	2206.6	3.1	4	1.90	3
	average	57.4	3.3	4.5	0.00	**3.2**	3.5	3.36	188.82	2745.84	2.67	3.5	61.48	3.9
25	0	68	2	2	0.00	2	2	2	179.67	3494.5	2	2	1.78	1
	1	84	5	7	0.01	5	5	5	339.16	3073.2	3	5	50.71	10
	2	50	3	3	0.00	3	3	3	174.30	4023	2	3	0.77	2
	3	80	4	5	0.01	4	4	4	143.16	2857.7	3.2	4	3.68	4
	4	66	4	5	0.01	4	4	4	91.22	3746	3	5	6.81	4
	5	72	4	4	0.01	4	4	4	641.87	2881.9	2	4	1.49	3
	6	74	4	5	0.01	4	5	4.7	143.24	3151	2.4	4	1.94	4
	7	62	3	3	0.01	3	3	3	206.71	4335.4	2	3	1.20	2
	8	64	5	8	0.00	5	5	5	312.33	6924.4	5.3	7	*	16
	9	66	5	5	0.01	5	5	5	88.56	3002	2	5	1.03	4
	average	68.6	3.9	4.7	0.00	3.9	4	3.97	232.02	3749.21	2.69	4.2	366.94	5
30	0	79	4	4	0.01	4	4	4	332.34	3352.9	2	4	1.72	3
	1	83	4	5	0.01	4	4	4	478.17	3788.4	3	4	2.09	4
	2	79	4	4	0.01	4	4	4	151.84	3443.9	2	4	1.85	3
	3	10 1	5	5	0.02	**4**	**4**	**4**	225.63	3081.2	3.1	5	2.73	4
	4	75	5	5	0.01	5	5	5	138.48	2871	2	5	1.32	4
	5	75	4	5	0.01	4	4	4	112.99	2760.8	3.2	4	7.37	4
	6	101	7	9	0.02	7	8	7.1	318.90	6483.3	4.5	8	*	24
	7	99	5	6	0.02	5	5	5	328.17	3714.5	3.1	6	*	9
	8	79	5	6	0.01	5	5	5	104.10	3448.6	3.1	5	9.67	9
	9	83	3	4	0.01	3	4	3.7	377.87	4289.5	2.6	4	10.33	3
	average	85.4	4.6	5.3	0.01	**4.5**	4.7	4.58	256.85	3723.41	2.86	4.9	723.71	6.7
35	0	94	4	5	0.02	4	4	4	314.51	5443.2	3.6	5	2.14	4
	1	92	3	4	0.01	3	3	3	186.81	2201.4	3	3	3.67	3
	2	106	3	3	0.01	3	3	3	210.59	2310.6	2	3	4.46	2
	3	112	7	8	0.03	7	8	7.7	1008.42	3700.9	2.3	8	25.03	18
	4	86	4	5	0.01	4	5	4.6	183.47	4079.1	2.8	5	1.69	4
	5	104	5	5	0.02	5	5	5	172.12	2920	2	5	2.90	4
	6	92	4	5	0.01	4	4	4	180.96	3580.1	3.1	5	3.04	4
	7	84	6	6	0.02	6	6	6	162.04	2642.7	2	6	6.20	9
	8	94	4	5	0.02	4	5	4.6	170.31	3602.9	2.7	5	2.14	4
	9	80	4	4	0.01	4	4	4	339.53	3098.7	2	4	1.68	3
	average	94.4	4.4	5	0.01	4.4	4.7	4.59	292.88	3357.96	2.55	4.9	5.29	5.5

Source: The authors (2020).

Table 4.2 – Results for small to medium-sized instances of type B with $n \leq 35$

Instance				GCA		TLMA						SWIA		
n	id	tasks	BKS	ub	t(s)	best	worst	avg	t(s)	iter	sol. count	ub	t(s)	iter
10	0	23	2	2	0.00	2	2	2	19.38	6005.7	2	2	0.41	1
	1	35	3	3	0.00	3	3	3	44.35	5571.3	2	3	0.32	2
	2	31	2	2	0.00	2	2	2	60.08	2338.7	2	2	0.30	1
	3	35	3	4	0.00	3	3	3	26.86	2944.9	3.5	4	0.28	3
	4	23	1	2	0.00	1	1	1	2.07	1000	26	1	2.96	1
	5	23	2	3	0.00	2	2	2	16	1507	3	3	0.16	2
	6	29	2	3	0.00	2	2	2	7.66	1505.1	3	2	0.27	2
	7	29	6	7	0.00	6	6	6	18.76	4528.5	3	6	484.46	15
	8	31	2	2	0.00	2	2	2	80.52	2007.3	2	2	0.28	1
	9	19	1	1	0.00	1	1	1	1.87	1000	200	1	0.05	1
	average	27.8	2.4	2.9	0.00	2.4	2.4	2.4	27.75	2840.85	24.65	2.6	48.95	2.9
15	0	48	2	2	0.00	2	2	2	35.89	2007.2	2	2	0.76	1
	1	48	3	4	0.00	3	3	3	58.02	3080.6	3.3	4	0.60	3
	2	48	2	3	0.00	2	2	2	27.71	1588.5	3	2	0.99	2
	3	52	3	3	0.00	3	3	3	92.78	4105.8	2	3	0.09	2
	4	40	2	2	0.00	2	2	2	49.07	2540.5	2	2	0.48	1
	5	42	2	3	0.00	2	2	2	25.45	1569.8	3	2	0.78	2
	6	38	3	4	0.00	3	4	3.3	37.28	4016.6	3.2	4	0.33	3
	7	46	6	7	0.00	6	6	6	31.90	4537.4	3	6	2557.68	15
	8	46	2	3	0.00	2	2	2	85.57	1955.4	3.1	3	41.52	2
	9	38	2	2	0.00	2	2	2	81.79	6603	2	2	*	1
	average	44.6	2.7	3.3	0.00	2.7	2.8	2.73	52.55	3200.48	2.66	3	620.40	3.2
20	0	63	4	5	0.01	4	4	4	233.32	3941.2	3.1	5	0.80	4
	1	63	6	7	0.01	6	7	6.2	71.75	3743.4	3.5	7	2.71	11
	2	59	3	3	0.00	3	3	3	126.33	4210.3	2	3	*	2
	3	55	2	3	0.00	2	3	2.7	57.17	3414.5	2.4	3	0.87	2
	4	51	4	5	0.00	4	4	4	84.16	3449	3	5	0.65	4
	5	61	4	4	0.01	4	4	4	211.50	2988.6	2	4	1.04	3
	6	55	4	5	0.01	**3**	4	3.9	89.76	2997.5	3.2	5	0.85	4
	7	63	3	4	0.00	3	4	3.5	164.78	4583.8	2.9	4	1.13	3
	8	43	3	3	0.00	3	3	3	44.23	2005.3	2	3	0.50	2
	9	61	7	10	0.01	7	7	7	46.63	4084.4	3.5	9	*	28
	average	57.4	4	4.9	0.00	**3.9**	4.3	4.13	112.96	3544.80	2.76	4.8	720.86	6.3
25	0	68	7	10	0.01	7	8	7.2	39.61	4297.6	3.4	8	*	27
	1	84	7	10	0.02	7	7	7	592.41	3890.4	3.6	9	*	23
	2	50	3	4	0.00	3	3	3	114.66	3151	3	3	0.97	3
	3	80	4	5	0.01	4	4	4	361.43	3980.3	3	4	1.64	4
	4	66	3	4	0.01	3	3	3	56.10	2330.3	3	4	*	3
	5	72	4	5	0.01	4	4	4	78.78	2704.9	3	5	1.42	4
	6	74	7	8	0.01	**6**	**6**	**6**	114.72	3052.2	3.8	8	*	18
	7	62	2	3	0.00	2	3	2.2	116.54	4068.9	2.8	2	306.99	2
	8	64	3	4	0.01	3	4	3.4	100.01	3220.2	2.8	4	1.17	3
	9	66	4	4	0.01	4	4	4	218.46	3905	2	4	1.15	3
	average	68.6	4.4	5.7	0.00	**4.3**	4.6	4.38	179.27	3460.08	3.04	5.1	1471.33	9
30	0	79	6	7	0.01	6	6	6	262.58	3113.7	3	7	3.79	11
	1	83	4	5	0.01	4	4	4	181.74	3504.2	3	4	1.89	4
	2	79	4	5	0.01	**3**	4	3.9	414.26	3970.1	3.2	4	1.66	4
	3	101	13	15	0.02	**12**	13	12.1	58.64	5197.7	3.3	14	*	75
	4	75	5	5	0.01	5	5	5	155.79	2924	2	5	1.30	4
	5	75	6	7	0.02	6	7	6.1	410.07	4634.4	3.4	7	47.08	11
	6	101	27	29	0.05	**26**	**26**	**26**	187.45	8789.9	4	28	*	299
	7	99	5	5	0.02	**4**	5	4.9	169.76	3402.2	2.2	5	2.68	4
	8	79	4	5	0.01	4	5	4.1	161.73	3129.2	3.3	5	1.42	4
	9	83	4	4	0.01	4	4	4	454.42	3192.2	2	4	1.71	3
	average	83.6	7.8	8.7	0.01	**7.4**	7.9	7.61	245.64	4125.76	2.94	8.3	726.15	41.9
35	0	94	8	10	0.03	**7**	8	7.1	163.07	4954.7	4.2	9	*	28
	1	92	4	5	0.01	4	5	4.9	454.16	6285.9	2.2	5	1.45	4
	2	106	4	5	0.01	4	4	4	915.33	4142.7	3.1	5	2.81	4
	3	112	7	10	0.02	7	7	7	352.27	3665.1	3.5	9	*	22
	4	86	3	4	0.01	3	4	3.1	136.41	3182.3	3.1	4	2.01	3
	5	104	7	8	0.03	7	7	7	1289.42	4026	3	8	*	18
	6	92	4	4	0.01	4	4	4	351.70	3451.5	2	4	2.36	3
	7	84	5	5	0.01	5	5	5	493.29	3422.3	2	5	1.26	4
	8	94	16	20	0.04	17	19	18	399.05	5028.1	3.2	18	*	128
	9	80	7	8	0.02	7	7	7	270.49	4342.8	3	8	*	18
	average	94.4	6.5	7.9	0.01	6.5	7	6.71	482.52	4250.14	2.93	7.5	1800.99	23.2

Source: The authors (2020).

Table 4.3 – Results for small to medium-sized instances of type C with $n \leq 35$

Instance				GCA		TLMA						SWIA		
n	id	tasks	BKS	ub	t(s)	best	worst	avg	t(s)	iter	sol. count	ub	t(s)	iter
10	0	27	3	3	0.00	3	3	3	5.22	1000	2.9	3	0.15	2
	1	27	10	10	0.00	10	10	10	74.27	15696.1	2	10	4.26	30
	2	27	9	9	0.00	9	9	9	7.77	3523.1	2	9	1.21	21
	3	27	10	10	0.00	10	10	10	156.98	27946.2	2	10	4.67	30
	4	27	10	**9**	0.00	**9**	**9**	**9**	8.91	3521.9	2	**9**	1.70	21
	5	27	2	3	0.00	2	2	2	12.18	1504	3	3	0.22	2
	6	27	3	3	0.00	3	3	3	12.85	4005	2	3	0.21	2
	7	27	5	6	0.00	5	5	5	4.65	2404	3	5	0.52	9
	8	27	5	5	0.00	5	5	5	3.80	1000	4.4	5	0.14	4
	9	27	2	2	0.00	2	2	2	18.29	6006	2	2	*	1
	average	27	5.9	6	0.00	**5.8**	**5.8**	**5.8**	30.49	6660.63	2.53	5.9	361.31	12.2
15	0	42	3	4	0.00	4	4	4	7.26	3396.1	2	4	0.33	3
	1	42	15	**14**	0.01	**14**	**14**	**14**	11.33	3867	2	**14**	97.09	63
	2	42	14	14	0.01	14	14	14	13.88	3867	2	14	500.15	63
	3	42	5	5	0.00	5	5	5	9.33	2424.7	2	5	0.35	4
	4	42	3	3	0.00	3	3	3	13.31	3003.2	2	3	0.38	2
	5	42	14	14	0.01	14	14	14	38.68	3556.8	2	14	79.40	63
	6	42	15	15	0.01	15	15	15	24.95	4084.2	2	15	535.56	69
	7	42	15	15	0.01	**14**	**14**	**14**	18.97	5644.3	3	15	*	69
	8	42	7	7	0.01	7	7	7	9.19	1000	2	7	0.78	11
	9	42	5	5	0.00	5	5	5	9.75	2866.5	2	5	0.26	4
	average	42	9.6	9.6	0.00	**9.5**	**9.5**	**9.5**	15.67	3370.98	2.1	9.6	481.43	35.1
20	0	57	-	3	0.00	3	3	3	18.15	4091.9	2	3	0.76	2
	1	57	-	10	0.01	10	10	10	80.73	1405.7	2	10	*	30
	2	57	-	19	0.02	19	19	19	21.45	3854.9	2	19	1412.78	116
	3	57	-	7	0.01	7	7	7	9.45	2681	2	7	1.91	11
	4	57	-	19	0.01	19	19	19	55.82	8552.9	2	19	1384.54	116
	5	57	-	6	0.01	5	5	5	31.63	4168.4	3	5	0.65	5
	6	57	-	8	0.01	7	7	7	80.62	5835.3	3.3	7	*	22
	7	57	-	8	0.01	7	7	7	20.50	4137.4	3	8	*	18
	8	57	-	7	0.01	7	7	7	14.78	2591.6	2	7	2.23	11
	9	57	-	3	0.00	3	3	3	19.84	1822	2	3	0.64	2
	average	57	9	9	0.00	**8.7**	**8.7**	**8.7**	35.30	3914.11	2.33	**8.8**	1360.42	33.3
25	0	72	-	4	0.01	4	4	4	47.78	3391.5	2	4	0.94	3
	1	72	-	3	0.01	3	3	3	44.63	1882.5	2	3	1.02	2
	2	72	-	8	0.01	8	8	8	18.79	2716.9	2	8	4.37	18
	3	72	-	24	0.02	24	24	24	103.76	3837	2	24	*	198
	4	72	-	4	0.01	4	4	4	30.97	3113.6	2	4	0.94	3
	5	72	-	4	0.01	3	3	3	26.27	2040	3	4	1.17	3
	6	72	-	25	0.02	25	25	25	170.26	7798.6	2	25	*	209
	7	72	-	25	0.03	24	24	24	78.36	7466	4	24	*	222
	8	72	-	6	0.01	6	6	6	48.20	3287.3	2	6	1279.34	9
	9	72	-	11	0.02	9	10	9.6	95.03	5897.1	3.8	11	*	34
	average	72	11.3	11.4	0.01	**11**	**11.1**	**11.06**	66.41	4143.05	2.48	11.3	1568.77	70.1
30	0	87	-	29	0.04	29	29	29	76.70	6077	2	29	*	286
	1	87	-	16	0.03	15	15	15	301.76	2819	3	15	*	71
	2	87	-	5	0.03	5	5	5	15.15	2011.3	2	5	1.22	4
	3	87	-	15	0.02	15	15	15	67.22	1629	2	15	*	69
	4	87	-	5	0.02	5	5	5	97.21	4828.7	2	5	1.19	4
	5	87	-	30	0.05	29	29	29	100.10	5459	3	30	*	315
	6	87	-	31	0.04	29	29	29	200.65	5274	3	30	*	316
	7	87	-	16	0.02	15	15	15	344.24	2819.9	3	15	*	70
	8	87	-	29	0.06	29	29	29	60.63	3795	2	29	*	286
	9	87	-	10	0.02	10	10	10	93.89	2738.2	2	10	*	30
	average	87	18.5	18.6	0.03	**18.1**	**18.1**	**18.1**	135.75	3745.11	2.4	**18.3**	2880.24	145.1
35	0	102	-	17	0.03	17	17	17	432.11	1884	2	17	*	91
	1	102	-	3	0.01	3	3	3	108.93	3033.2	2	3	2.16	2
	2	102	-	12	0.04	12	12	12	25.68	2741.1	2	12	*	45
	3	102	-	8	0.04	8	8	8	253.92	12293.7	2	8	9.28	18
	4	102	-	12	0.03	12	12	12	62.55	2828.8	2	12	*	45
	5	102	-	18	0.07	17	17	17	315.73	3014.2	3	18	*	108
	6	102	-	14	0.03	12	12	12	197.33	3490.5	3	12	*	47
	7	102	-	13	0.03	12	13	12.9	548.66	6523.3	2.1	13	*	50
	8	102	-	34	0.06	34	34	34	378.72	4065	2	34	*	408
	9	102	-	4	0.02	4	4	4	89.57	3247.9	2	4	1.75	3
	average	102	13.5	13.5	0.03	**13.1**	**13.2**	**13.19**	241.32	4312.17	2.21	**13.3**	2521.31	81.7

Source: The authors (2020).

Table 4.4 – Results for medium to large-sized instances of types A, B and C with $n \geq 40$

Instances				GCA		TLMA						SWIA		
type	n	tasks	BKS	ub	t(s)	best	worst	avg	t(s)	iter	sol. count	ub	t(s)	iter
A	40	121.2	6.7	8.1	0.03	6.7	7	6.86	684.09	4012.27	3.27	7.4	1814.03	12.6
	50	145.2	7	7.5	0.06	**6.6**	7	**6.79**	1033.48	3257.55	2.41	7.1	1868.05	14.5
	60	171.2	9.8	11.5	0.10	9.8	10.3	9.94	1242.81	3335.48	2.64	10.6	2902.46	22
	70	201.2	8.5	9.5	0.15	**8**	8.5	**8.38**	1250.81	3066.74	2.40	9.1	2905.17	17.6
	80	235	11.2	12	0.29	**10.8**	11.5	**11.17**	1704.58	1693.05	1.23	11.5	2769.61	25.3
	90	270.6	14.7	15.8	0.44	15	15.5	15.35	1800	582.64	0.28	15.1	3288.83	38.5
	100	304	17.2	18.5	0.59	17.8	18.5	18.14	1800	622.99	0.27	17.7	3529.27	43.8
B	40	121.2	7.1	8.1	0.03	**6.9**	7.2	**7.03**	515.23	3821.35	2.86	7.5	1485.51	23.3
	50	145.2	7.1	9	0.05	**6.9**	7.6	7.22	792.46	3988.82	2.76	8.1	1120.08	25.9
	60	171.2	13.7	16.2	0.11	**13.5**	14.3	13.76	1118.59	4099.17	2.67	15.2	2694.90	110.2
	70	201.2	12.7	14.5	0.16	**12.4**	13.2	**12.69**	1354.77	3458.40	2.40	13.8	3600	87.9
	80	235	15.5	17.1	0.28	**15**	**15.4**	**15.21**	1494.97	2957.64	1.78	16.6	3600	110.1
	90	270.6	14.1	16.4	0.44	15	15.4	15.79	1794.21	1232.48	0.76	15.3	3284.06	39.1
	100	304	15.2	16.4	0.54	**14.9**	15.3	15.01	1798.67	1421.46	0.78	15.6	3036.99	50.2
C	40	117	16.2	16.8	0.06	**16.1**	16.4	16.28	257.51	4507.28	2.59	16.8	2170.19	69.9
	50	147	18.1	18.3	0.12	**18**	18.1	18.07	723.27	6520.18	1.92	**17.8**	2188.92	72.3
	60	177	39.6	**39.4**	0.35	**39.1**	**39.3**	**39.13**	1181.31	3774.87	1.65	**39.4**	3288.85	210.5
	70	207	27.9	28.5	0.43	**27.8**	28.4	28.16	1287.77	4067.85	1.67	28.2	2928.82	120
	80	237	41.5	42.4	1.04	**41.4**	41.5	41.42	1358.22	3179.61	1.69	42.2	3327.92	220.4
	90	267	51.1	51.1	1.31	**50.7**	**50.8**	**50.79**	1558.95	2251.19	1.08	**50.8**	3600	244.6
	100	297	53.3	54	2.30	53.8	53.9	53.82	1642.90	1792.64	0.76	53.9	3600	266

Source: The authors (2020).

5 NEGATIVE RESULTS

In this section we discuss some of the approaches that did not turn out as successful as expected, but can provide insight for further research.

5.1 Can a route be considered "good enough"?

In the TLMA, a situation would often occur where feasible routes would be found for only a subset of teams instead of the whole set. Consider as an example a problem with 3 teams $k \in K$ and $T = 8$ units of time for the completion of requested activities. Let $\{7.8, 7.5, 8.2\}$ be the corresponding costs (in units of time) of each route. Routes for teams $\{1, 2\}$ are feasible. Team 3's route, however, violates daily available time by 0.2. When reaching this type of scenario, we decided to investigate the impact of considering the feasible routes as finished and only trying to solve the remaining ones on further iterations. This was achieved by having the first-level scheduling that lead to feasible routes *fixed* for the 'finished' teams in the following iterations of the algorithm.

The *fixing* of one or more schedules on the first-level can have a very limiting impact on the route building possibility for remaining teams on the second-level. This occurs due to the possibility of one or more tasks in the *fixed* teams being fulfilled representing a requirement for the remaining tasks in the available pool. If that was the case, affected tasks still in the pool would have a set minimum start time for every new attempt at routing the remaining teams. The latter consideration led to the following question: *when* is a route good enough to be fixed? To try answering that question we attempted using the concept of good route quality being tied to low idle time for a team. Let $R^k = \{\{i,a\} : \{i,a\}$ is executed by team k on the current day in the algorithms execution$\}$. Let Sol^{kh} be the an array representation of the scheduling designed for team k on day h. Values in Sol^{kh} represent the order in which activities are performed and are stored in the form $(i, a, start, end)$, where i represents the customer team k is located at, a is the task being performed, and $start$ and end the time the task starts and ends, respectively. Within this solution representation, the idle time of a teams route is defined by the following equation: $\sum\limits_{i \in 1,\ldots,|Sol^{kh}|-1} Sol^{kh}[i+1][2] - Sol^{kh}[i][3] + d_{Sol^{kh}[i][0] Sol^{kh}[i+i][0]}$. According to this equation, idle time will not exist in a route if the team did not have to wait in order to start performing the next task. Otherwise, it will exist if between any pair of visits assigned to that day, the team took longer than the travel time between this pair of customers to start performing the requested task.

Experiments were conducted by allowing a teams' route to be fixed if the route used at least 90% of the daily available time and had low idle time. Acceptance values for idle time were experimented ranging from 0 to 0.5 time units. If a route for a team was considered *good enough*, the variables corresponding to the appointments for the team on the respective day were set to 1 on the model execution throughout the following iterations of the algorithm. Overall, this approach did not provide improvement in regards to solution quality.

5.2 Dispatching Rules

In agreement with discussions seen in the literature, the choice of Dispatching Rule (DR) for each of the proposed algorithms in this document was shown to have a significant impact on solution quality. Our experience can also be used to validate the concept that the dispatching rule used when solving optimization problems must be carefully selected for the problem and the algorithm being used.

Our choice of dispatching rules were mostly based on the intrinsic aspects of the MWS-RPDT: the need to visit customers and the characteristics of the provided services. When dealing with customer visits, arguably the most important aspect to note is the distance between those. Regarding the provided services, one can derive a few important aspects, such as: amount of tasks, amount of dependencies between these tasks, amount of technicians with the needed skill set to perform these tasks, and etc.

In an attempt to encompass these characteristics, we experimented with the dispatching rules discussed next. Let $i \in V$ be the customer team k is currently stationed and R be the set of $\{j,a\}$ customer-task pairs available at any given point in the algorithms execution. The simplest DR implementation we experimented considered only travel distance between customers, according to the following equation: (i) $\min_{\{j,a\} \in R} d_{ij}$. Based on this simplistic concept, we also added team task-performing times and distance to the depot 1, leading to (ii) $\min_{\{j,a\} \in R} d_{ij} + t^k_{ja}$ and (iii) $\min_{\{j,a\} \in R} d_{ij} + t^k_{ja} + d_{j1}$, respectively.

Prioritizing service characteristics instead of travel distance, we experimented with two DRs. Let σ be the set of complete customer-task pairs and $L_{ia} = \{\{i,b\} : (b,a) \in A_{s^i}, \{i,b\} \notin R,$ $\{i,b\} \notin \sigma\}$ be the set of requested activities that depend on task $\{i,a\}$ that have not been fulfilled and are still unavailable at any given point in the algorithms execution. Activities were selected according to the following equations: $(iv)\ \min_{\{i,a\}\in R} |L_{ia}|, \{i,a\} \in R$ and $(v)\ \max|L_{ia}|$. These DRs intended to prioritizing the completion of tasks that had the least/most dependencies unfulfilled. Their performance was shown to be inferior to the distance-concerning ones. This observation might indicate that travel distance between customers' significance is high on most of our instances, and therefore, can not be underestimated.

The use of random and/or pseudo-randomness in combinatorial optimization is vast and has been shown to be a characteristic that can lead to good quality solutions (HOLTHAUS; RAJENDRAN, 1997). In light of this fact, we experimented with another simplistic DR that would simply select the next task from the pool of available activities with pseudo-random chance (vi). In our case, however, it did not lead to significant improvement with reasonable frequency to justify its use. Finally, an experiment was made by allowing the algorithms choice of DR to be made with a pseudo-random chance. At every step of the GCA (and its adapted version used in the TLMA), a DR would be chosen if pseudo-random chance from sets of 3 different DRs. Overall, the best combination results were found by a set composed of *ii*, *v* and *vi*. These results were, however, also not an improvement upon making use of a single DR.

5.3 Weighted function impact of both algorithms

Both the original model of Pereira, Alves and Moreira (2020) and the simplified one proposed in this document share an objective function (3.1) that aims to minimize the amount of periods expended for the completion of every requested task. The use of this objective function, however, was shown to be insufficient in some situations. In this section, we will discuss when and how that situation arises.

When dealing with the TLMA, the results found by this function are tied to the quality of the existent upper bound for the problem. The initial upper bound serves as very strong guideline that ensures the simplified model on the first level is guaranteed to return a scheduling for the available tasks that can be successfully routed on the second. What we observed throughout experimenting was that the combination of an inferior quality upper bound with the added cuts in (3.17) would lead to a situation where the amount of tasks scheduled for the first day would decrease. These tasks would be pushed on to a later day and a successful routing on the first day would eventually be found with very few activities. Consequently, a lot of available time would remain unused on that day. This is not ideal and often lead to the discovery of a final solution that would at most match, not improve, the incumbent solution. Therefore, a weak upper bound combined with this objective function would often lead to low quality solutions.

In order to 'fix' this occurrence we introduced a different objective function:

$$\min p - \frac{\sum_{k=1}^{K} \sum_{\{i,a\} \in R} y_{ia}^{k1}}{|R|} \tag{5.1}$$

This objective function provides the model with an incentive to assign tasks on the first day of the planning horizon instead of the following days by giving the variables corresponding to the first day's assignments a negative cost. This new objective function, however, comes with a significant increase in computational cost. The use of this function on the model (3.1)-(3.10), even when used as a local search operator, was shown to be impractical due to the computational effort it required. On the simplified model, however, the increase in computational cost does not make it unusable. Thus, when Algorithm 1 reaches a situation where too few tasks are being assigned on the first day, its use was able to improve solution quality and overcome the effect of the added cuts. Adapting the algorithm to make use of this objective function when deemed necessary showed significant improvement in solution quality. However, as we previously noted, an increase in computational cost did also occur. For that reason, the use of this function is advised when coupled with a dynamic approach that manages a good ratio between solution quality and computational effort.

6 CONCLUSION

Logistics problems are heavily present in both enterprise and academic contexts. In many industries, they lie at the core of most business models and have a significant impact on a companies success. Therefore, researchers have also thoroughly studied these types of problems in attempts to present innovative solution methods that can deliver high-quality and applicable solutions. In the last two decades, a class of combinatorial optimization problems that incorporate these aspects, named Workforce Scheduling and Routing Problems (WSRP), have been given significant attention from researchers. In a general WSRP, clients request services from a provider that must service every customer through mobile teams. Client's requests must be scheduled for available teams and visits must be effectively routed. In this document, we studied a recently proposed WSRP named the Multiperiod Workforce Scheduling and Routing Problem with Dependent Tasks (MWSRPDT). In the MWSRPDT, dependencies between activities are also present, making the problem significantly harder to solve.

Following the literature on combinatorial optimization problems, WSRPs are usually solved by means of exact, heuristic, or, in a more recent trend, hybrid algorithms. Due to their complexity, the employment of purely exact algorithms for WSRPs is quite difficult. Therefore, heuristics and hybrid algorithms that attempt to use both exact and heuristic concepts have emerged as very desirable solution methods for these types of problems. In this document, we present two hybrid exact-heuristic algorithms, the TLMA and the SWIA, for the MWSRPDT that make use of a mathematical model for the problem recently proposed in (PEREIRA; ALVES; MOREIRA, 2020). The first, uses a simplified version of the model to perform client scheduling followed by a routing procedure imbued with a local search technique applied to precedence-free spaces that plans routes for every team. Perhaps the greatest contribution of this document is the identification of precedence-free spaces. In those, the employment of local search techniques is available with no concern about ruining the solution's feasibility. The second uses the original model as a local search operator in subsequent small windows of an existing solution obtained by a greedy algorithm, returning an optimal solution for that window.

While the SWIA did not manage to obtain significant enough results to compete with results in the literature, the TLMA managed to improve the Best Known Solution values for 12 individual instances and 27 averages with reasonable computational effort. Once the 80 client threshold is met, however, the algorithm loses strength. The algorithm can be seen as a middle-ground between exact and heuristic solution methods that can deliver high-quality solutions with low computational effort on small to medium-sized instances. However, the use of the simplified version of the model is accompanied by computational time-related scalability issues. Therefore, its use is advised mainly on instances with less than or equal to 80 customers.

For future research, we intend to improve the simplified model further in an attempt to lower its required computational effort on large instances and implement more robust heuristic techniques on precedence-free spaces.

BIBLIOGRAPHY

ABDULKADER, M. M. S.; GAJPAL, Y.; ELMEKKAWY, T. Y. Hybridized ant colony algorithm for the multi compartment vehicle routing problem. **Applied Soft Computing**, v. 37, p. 196–203, 2015.

ALVAREZ, A.; MUNARI, P. An exact hybrid method for the vehicle routing problem with time windows and multiple deliverymen. **Computers & Operations Research**, v. 83, p. 1–12, 2017.

AMIN, G. R.; EL-BOURI, A. A minimax linear programming model for dispatching rule selection. **Computers & Industrial Engineering**, v. 121, p. 27–35, 2018.

BELL, J. E.; MCMULLEN, P. R. Ant colony optimization techniques for the vehicle routing problem. **Advanced Engineering Informatics**, v. 18, n. 1, p. 41–48, 2004.

BENDERS, J. Partitioning procedures for solving mixed-variables programming problems. **Numerische Mathematik**, v. 4, n. 1, p. 238–252, 1962.

BLACKSTONE, J. H.; PHILLIPS, D. T.; HOGG, G. L. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. **International Journal of Production Research**, v. 20, n. 1, p. 27–45, 1981.

BLUM, C. et al. Hybrid metaheuristics in combinatorial optimization: A survey. **Applied Soft Computing**, v. 11, n. 6, p. 4135–4151, 2011.

BOSTEL, N. et al. Multiperiod planning and routing on a rolling horizon for field force optimization logistics. In: GOLDEN, B.; RAGHAVAN, S.; WASIL, E. (Ed.). **The Vehicle Routing Problem: Latest Advances and New Challenges**. Boston, MA: Springer, 2008. v. 43, p. 503–525.

BOWMAN, E. H. The schedule-sequencing problem. **Operations Research**, v. 7, n. 5, p. 621–624, 1959.

BRANDIMARTE, P. Routing and scheduling in a flexible job shop by tabu search. **Annals of Operations Research**, v. 41, p. 157–183, 1993.

BRANKE, J.; HILDEBRANDT, T.; SCHOLZ-REITER, B. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. **Evolutionary Computation**, v. 23, p. 249–277, 2015.

BREDSTRöM, D.; RöNNQVIST, M. A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. **SSRN Electronic Journal**, 2007.

BRUCKER, P.; JURISCH, B.; SIEVERS, B. A branch and bound algorithm for the job-shop scheduling problem. **Discrete Applied Mathematics**, v. 49, p. 107–127, 1994.

CASTILLO-SALAZAR, J. A.; LANDA-SILVA, D.; QU, R. Workforce scheduling and routing problems: literature survey and computational study. **Annals of Operations Research**, v. 239, n. 1, p. 39–67, 2016.

CHANG, Y.-L.; SUEYOSHI, T.; SULLIVAN, R. S. Ranking dispatching rules by data envelopment analysis in a job shop environment. **IIE Transactions**, Taylor & Francis, v. 28, n. 8, p. 631–642, 1996.

CHAUDHRY, I. A.; KHAN, A. A. A research survey: review of flexible job shop scheduling techniques. **International Transactions in Operational Research**, v. 23, n. 3, p. 551–591, 2016.

CHEN, X.; THOMAS, B. W.; HEWITT, M. The technician routing problem with experience-based service times. **Omega**, v. 61, p. 49–61, 2016.

CHIANG, W.-C.; RUSSEL, R. A. Simulated annealing metaheuristics for the vehicle routing problem with time windows. **Annals of Operations Research**, v. 630, n. 1, p. 3–27, 1996.

CHRISTOFIDES, N.; EILON, S. An algorithm for the vehicle routing dispatching problem. **Operations Research Quaterly**, v. 20, p. 309–318, 1969.

CLARKE, G.; WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. **Operations Research**, v. 12, n. 4, p. 568–581, 1964.

CORDEAU, J.-F.; LAPORTE, G.; MERCIER, A. A unified tabu search heuristic for vehicle routing problems with time windows. **Journal of the Operational Research Society**, v. 52, n. 8, p. 928–936, 2001.

CORDEAU, J.-F. et al. Scheduling technicians and tasks in a telecommunications company. **Journal of Scheduling**, v. 13, n. 4, p. 393–409, 2010.

CROES, G. A. A method for solving traveling-salesman problems. **Operations Research**, v. 6, n. 6, p. 791–812, 1958.

CRUZ, J. C. et al. Rich vehicle routing problem: Survey. **ACM Computing Surveys**, v. 47, p. 1–28, 2014.

DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. **Management Science**, v. 6, n. 1, p. 80–91, 1959.

DELL'AMICO, M.; TRUBIAN, M. Applying tabu search to the job-shop scheduling problem. **Annals of Operations Research**, v. 41, n. 3, p. 234–252, 1993.

DESROCHERS, M.; DESROSIERS, J.; SOLOMON, M. A new optimization algorithm for the vehicle routing problem with time windows. **Operations Research**, v. 40, n. 2, p. 342–354, 1992.

DOHN, A.; KOLIND, E.; CLAUSEN, J. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. **Computers & Operations Research**, v. 36, n. 4, p. 1145–1157, 2009.

DOMINIC, P.; KALIYAMOORTHY, S.; KUMAR, M. Efficient dispatching rules for dynamic job shop scheduling. **The International Journal of Advanced Manufacturing Technology**, v. 24, n. 1-2, p. 70–75, 2004.

DORIGO, M.; COLORNI, A.; MANIEZZO, V. Distributed optimization by ant colonies. **European Conference on Artifical Life**, v. 91, p. 134–142, 1991.

DREXL, M. Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints. **Transportation Science**, v. 46, n. 3, p. 297–316, 2012.

ESCOBAR, J. W.; LINFATI, R.; BALDOQUIN, M. G. A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. **Journal of Heuristics**, v. 20, n. 5, p. 483–509, 2014.

FISCHETTI, M.; LODI, A. Mathematical programming. **Operations Research**, v. 98, n. 1-3, p. 23–47, 2003.

FISHER, M. L.; JAIKUMAR, R. A generalized assignment heuristic for vehicle routing. **Networks**, v. 11, n. 2, p. 109–124, 1981.

FUKASAWA, R. et al. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. **Mathenatical Programming**, v. 106, n. 3, p. 491–511, 2006.

GENDREAU, M.; HERTZ, A.; LAPORTE, G. A tabu search heuristic for the vehicle routing problem. **Maganement Science**, v. 40, n. 10, p. 1276–1290, 1994.

GLOVER, F. Tabu search–part i. **ORSA Journal on Computing**, v. 1, n. 3, p. 190–206, 1989.

GLOVER, F.; KOCHENBERGER, G. A. **Handbook of Metaheuristics**. 1. ed. [S.l.]: Boston : Kluwer Academic Publishers, 2003.

GOEL, A.; MEISEL, F. Workforce routing and scheduling for electricity network maintenance with downtime minimization. **European Journal of Operational Research**, v. 231, n. 1, p. 210–228, 2013.

GOLDEN, B. L.; RAGHAVAN, S.; WASIL, E. **The Vehicle Routing Problem: Latest Advances and New Challenges**. 1. ed. [S.l.]: Springer Verlag NY, 2008.

GONCALVES, J. F.; MENDES, J. d. M.; RESENDE, M. G. C. A hybrid genetic algorithm for the job shop scheduling problem. **European Journal of Operational Research**, v. 167, n. 1, p. 77–95, 2005.

GRABOWSKI, J.; NOWICKI, E.; ZDRZAłKA, S. A block approach for single-machine scheduling with release dates and due dates. **European Journal of Operational Research**, v. 26, n. 2, p. 278–285, 1986.

GUASTAROBA, G.; CôTé, J.-F.; COELHO, L. The multi-period workforce scheduling and routing problem. **Omega**, p. 102302, 2020.

GUTIERREZ, A. et al. A hybrid metaheuristic algorithm for the vehicle routing problem with stochastic demands. **Computers & Operations Research**, v. 99, p. 135–147, 2018.

GUTIN, G.; YEO, A.; ZVEROVICH, A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. **Discrete Applied Mathematics**, v. 117, n. 1-3, p. 81–86, 2002.

HIERMANN, G. et al. Routing a mix of conventional, plug-in hybrid, and electric vehicles. **European Journal of Operational Research**, v. 272, n. 1, p. 235–248, 2019.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1992.

HOLTHAUS, O.; RAJENDRAN, C. Efficient dispatching rules for scheduling in a job shop. **International Journal of Production Economics**, v. 48, n. 1, p. 87–105, 1997.

INGIMUNDARDOTTIR, H.; RUNARSSON, T. P. Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. **Journal of Scheduling**, v. 21, p. 413–428, 2018.

IRNICH, S.; SCHNEIDER, M.; VIGO, D. Four variants of the vehicle routing problem. In: TOTH, P.; VIGO, D. (Ed.). **Vehicle Routing: Problems, Methods and Applications**. [S.l.]: SIAM, 2014. chap. 9, p. 241–271.

IRNICH, S.; TOTH, P.; VIGO, D. The family of vehicle routing problems. In: TOTH, P.; VIGO, D. (Ed.). **Vehicle Routing: Problems, Methods and Applications**. [S.l.]: SIAM, 2014. chap. 5, p. 1–33.

JONES, P. The evolution of urban mobility: The interplay of academic and policy perspectives. **IATSS Research**, v. 38, n. 1, p. 7–13, 2014.

JUN, S.; LEE, S.; CHUN, H. Learning dispatching rules using random forest in flexible job shop scheduling problems. **International Journal of Production Research**, Taylor Francis, v. 57, n. 10, p. 3290–3310, 2019.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: **Proceedings of ICNN'95 - International Conference on Neural Networks**. [S.l.]: IEEE, 1995.

Khalfay, A.; Crispin, A.; Crockett, K. A review of technician and task scheduling problems, datasets and solution approaches. In: **2017 Intelligent Systems Conference (IntelliSys)**. [S.l.: s.n.], 2017. p. 288–296.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, v. 220, n. 4598, p. 671–680, 1983.

KOHL, N. et al. 2path cuts for the vehicle routing problem with time windows. **Transportation Science**, v. 33, p. 101–116, 1999.

KOVACS, A. A. et al. Adaptive large neighborhood search for service technician routing and scheduling problems. **Journal of Scheduling**, v. 15, n. 5, p. 579–600, 2012.

KU, W.-Y.; BECK, J. C. Mixed integer programming models for job shop scheduling: A computational analysis. **Computers & Operations Research**, v. 73, 2016.

LAPORTE, G. The vehicle routing problem: An overview of exact and approximate algorithms. **European Journal of Operational Research**, v. 59, p. 345–358, 1992.

LAPORTE, G.; NOBERT, Y. A branch and bound algorithm for the capacitated vehicle routing problem. **Operations-Research-Spektrum**, v. 5, n. 2, p. 77–85, 1983.

LENSTRA, J.; KAN, A. R.; BRUCKER, P. Complexity of machine scheduling problems. In: HAMMER, P. et al. (Ed.). **Studies in Integer Programming**. [S.l.]: Elsevier, 1977, (Annals of Discrete Mathematics, v. 1). p. 343–362.

LENSTRA, J. K.; KAN, A. H. G. R. Complexity of scheduling under precedence constraints. **Operations Research**, v. 26, n. 1, p. 22–35, 1978.

LENSTRA, J. K.; KAN, A. H. G. R. Complexity of vehicle routing and scheduling problems. **Networks**, v. 11, n. 2, p. 221–227, 1981.

LI, J.-Q.; PAN, Q.-K.; SUGANTHAN, P. N. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, v. 52, p. 683–697, 2010.

LI, X.; OLAFSSON, S. Discovering dispatching rules using data mining. **Journal of Scheduling**, v. 8, n. 6, p. 515–527, 2005.

MANIEZZO, V.; STüTZLE, T.; VOSS, S. **Matheuristics – Hybridizing Metaheuristics and Mathematical Programming**. [S.l.: s.n.], 2010.

MANNE, A. S. On the job-shop scheduling problem. **Operations Research**, v. 8, n. 2, p. 219–223, 1960.

MASTROLILLI, M.; GAMBARDELLA, L. M. Effective neighborhood functions for the flexible job shop problem. **Journal of Scheduling - SCHEDULING**, v. 3, 2000.

MILLER, H. E. Personnel scheduling in public systems: a survey. **Socio-Economic Planning Sciences**, v. 10, n. 6, p. 241–249, 1976.

MITTEN, L. G. Branch-and-bound methods: General formulation and properties. **Operations Research**, v. 18, n. 1, p. 24–34, 1970.

MLADENOVIć, N.; HANSEN, P. Variable neighborhood search. **Computers & Operations Research**, v. 24, n. 11, p. 1097–1100, 1997.

MOLE, R. H.; JAMESON, S. R. A sequential route-building algorithm employing a generalised savings criterion. **Journal of the Operational Research Society**, v. 27, n. 2, p. 503–511, 1976.

MORENO, A.; MUNARI, P.; ALEM, D. A branch-and-benders-cut algorithm for the crew scheduling and routing problem in road restoration. **European Journal of Operational Research**, v. 275, n. 1, p. 16–34, 2019.

MOUSSAVI, S.; MAHDJOUB, M.; GRUNDER, O. A matheuristic approach to the integration of worker assignment and vehicle routing problems: Application to home healthcare scheduling. **Expert Systems with Applications**, v. 125, p. 317–332, 2019.

NGUYEN, S.; ZHANG, M.; TAN, K. C. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. **IEEE Transactions on Cybernetics**, v. 47, n. 9, p. 2951–2965, 2016.

NOWICKI, E.; SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. **Maganement Science**, v. 42, n. 6, p. 797–813, 1996.

OPTIMIZATION, L. G. **Gurobi Optimizer Reference Manual**. 2020. Available at: <http://www.gurobi.com>.

PECIN, D. et al. Improved branch-cut-and-price for capacitated vehicle routing. **Mathematical Programming Computation**, v. 9, n. 1, p. 61–100, 2017.

PENNA, P. H. V. et al. A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. **Annals of Operations Research**, v. 273, n. 1-3, p. 5–74, 2019.

PEREIRA, D. L.; ALVES, J. C.; MOREIRA, M. C. de O. A multiperiod workforce scheduling and routing problem with dependent tasks. **Computers & Operations Research**, v. 118, p. 104930, 2020.

PESSOA, A.; ARAGAO, M. de; UCHOA, E. Robust branch-cut-and-price algorithms for vehicle routing problems. In: GOLDEN, B.; RAGHAVAN, S.; WASIL, E. (Ed.). **The Vehicle Routing Problem: Latest Advances and New Challenges**. Boston, MA: Springer, 2008.

PESSOA, A.; SADYKOV, R.; UCHOA, E. Enhanced branh-cut-and-price algorithm for heterogeneous fleet vehicle routing problems. **European Journal of Operational Research**, v. 270, p. 530–543, 2018.

PILLAC, V.; GUÉRET, C.; MEDAGLIA, A. L. A parallel matheuristic for the technician routing and scheduling problem. **Optimization Letters**, v. 7, n. 7, p. 1525–1535, 2013.

PILLAC, V.; GUéRET, C.; MEDAGLIA, A. On the technician routing and scheduling problem. In: **Proceedings of the 9th Metaheuristics Conference**. [S.l.: s.n.], 2011. p. 675–678.

PISINGER, D.; ROPKE, S. A general heuristic for vehicle routing problems. **Computers & Operations Research**, v. 34, p. 2403–2435, 2007.

PRINS, C. A simple and effective evolutionary algorithm for the vehicle routing problem. **Computers & Operations Research**, v. 31, n. 12, p. 1985–2002, 2004.

PRIORE, P. et al. Learning-based scheduling of flexible manufacturing systems using ensemble methods. **Computers & Industrial Engineering**, v. 126, p. 282 – 291, 2018.

RASMUSSEN, M. S. et al. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. **European Journal of Operational Research**, v. 219, n. 3, p. 598–610, 2012.

REED, M.; YIANNAKOU, A.; EVERING, R. An ant colony algorithm for the multi-compartment vehicle routing problem. **Applied Soft Computing**, v. 15, p. 169–176, 2014.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Transportation Science**, v. 40, p. 455–472, 2006.

ROSSUM, G. V.; JR, F. L. D. **Python tutorial**. Amsterdam: Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

ROY, B.; SUSSMANN, B. Les problemes d'ordonnancement avec contraintes disjonctives. **Note ds**, SEMA Montrouge, v. 9, 1964.

SANTIS, R. D. et al. An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses. **European Journal of Operation Research**, v. 267, n. 1, p. 120–137, 2018.

SHI, Y.; BOUDOUH, T.; GRUNDER, O. A robust optimization for a home health care routing and scheduling problem with consideration of uncertain travel and service times. **Transportation Research Part E: Logistics and Transportation Review**, v. 128, p. 52–95, 2019.

SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. **Operations Research**, v. 35, n. 2, p. 254–265, 1987.

SUNDAR, S. et al. A hybrid artificial bee colony algorithm for the job-shop scheduline problem with no-wait constraint. **Soft Computing**, v. 21, p. 1193–1202, 2017.

TALBI, E.-G.; EL-GHAZALI. **Metaheuristics: From Design to Implementation**. [S.l.: s.n.], 2009.

TAY, J. C.; HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. **Computers & Industrial Engineering**, v. 54, n. 3, p. 453–473, 2008.

TOTH, P.; VIGO, D. **Vehicle Routing: Problems, Methods and Applications**. 2. ed. [S.l.]: SIAM, 2014.

VIDAL, T. et al. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. **Operations Research**, v. 60, n. 3, p. 611–624, 2012.

VIDAL, T. et al. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. **European Journal of Operational Research**, v. 231, n. 1, p. 1–21, 2013.

VIDAL, T. et al. A unified solution framework for multi-attribute vehicle routing problems. **European Journal of Operational Research**, v. 234, n. 3, p. 658–673, 2014.

VIDAL, T.; LAPORTE, G.; MATL, P. A concise guide to existing and emerging vehicle routing problem variants. **European Journal of Operational Research**, v. 286, n. 2, p. 401–416, 2020.

WAGNER, H. M. An integer linear-programming model for machine scheduling. **Naval Research Logistics Quarterly**, v. 6, n. 2, p. 131–140, 1959.

WEI, L. et al. A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. **European Journal of Operational Research**, v. 265, n. 3, p. 843–859, 2018.

WOLSEY, L. **Integer Programming**. [S.l.]: Wiley, 1998. (Wiley Series in Discrete Mathematics and Optimization).

XIE, F.; POTTS, C. N.; BEKTAŞ, T. Iterated local search for workforce scheduling and routing problems. **Journal of Heuristics**, v. 23, n. 6, p. 471–500, 2017.

XING, L.-N. et al. A knowledge-based ant colony optimization for flexible job shop scheduling problems. **Applied Soft Computing**, v. 10, p. 888–896, 2010.

XY, L.; LIANG, G. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. **International Journal of Production Economics**, v. 174, p. 93–110, 2016.

ZAMORANO, E.; STOLLETZ, R. Branch-and-price approaches for the multiperiod technician routing and scheduling problem. **European Journal of Operational Research**, v. 257, n. 1, p. 55–68, 2017.

ZHAN, Y.; WAN, G. Vehicle routing and appointment scheduling with team assignment for home services. **Computers & Operations Research**, v. 100, p. 1–11, 2018.

ÇAKıRGIL, S.; YüCEL, E.; KUYZU, G. An integrated solution approach for multi-objective, multi-skill workforce scheduling and routing problems. **Computers & Operations Research**, v. 118, p. 104908, 2020.