UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Emiliana Mara Lopes Simões

A MATHEURISTIC ALGORITHM FOR THE MULTIPLE-DEPOT VEHICLE AND CREW SCHEDULING PROBLEM

Belo Horizonte
2022

Emiliana Mara Lopes Simões

A MATHEURISTIC ALGORITHM FOR THE MULTIPLE-DEPOT VEHICLE AND CREW SCHEDULING PROBLEM

Final Version

Dissertation presented to the Graduate Program in Electrical Engineering of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

Advisor: Lucas de Souza Batista
Co-Advisor: Marcone Jamilson Freitas Souza

Belo Horizonte
2022

Simões, Emiliana Mara Lopes.
S593m A matheuristic algorithm for the multiple-depot vehicle and crew scheduling problem [recurso eletrônico] / Emiliana Mara Lopes Simões. - 2022.
1 recurso online (92 f. : il., color.) : pdf.

Orientador: Lucas de Souza Batista.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 86-92.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Algoritmos - Teses. 3. Modelos matemáticos - Teses. 4. Otimização. 5. Transportes coletivos - Teses. I. Batista, Lucas de Souza. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pelo bibliotecário Reginaldo César Vital dos Santos - CRB-6/2165
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG.

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

"A MATHEURISTIC ALGORITHM FOR THE MULTIPLE-DEPOT VEHICLE AND CREW SCHEDULING PROBLEM"

EMILIANA MARA LOPES SIMÕES

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Doutor em Engenharia Elétrica.

Aprovada em 13 de julho de 2022. Por:

Prof. Dr. Lucas de Souza Batista
DEE (UFMG) - Orientador

Prof. Dr. Marcone Jamilson Freitas Souza
Departamento de Computação (UFOP)

Prof. Dr. Geraldo Robson Mateus
DCC (UFMG)

Prof. Dr. Mauricio Cardoso de Souza
DEP (UFMG)

Prof. Dr. Claudio Barbieri da Cunha
Departamento de Engenharia de Transportes (EPUSP)

Prof. Dr. André Luiz Maravilha Silva
Departamento de Informática, Gestão e Design (CEFET-MG)

Documento assinado eletronicamente por **Lucas de Souza Batista**, **Professor do Magistério Superior**, em 13/07/2022, às 19:27, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1576924** e o código CRC **8F165C68**.

Dedico este trabalho ao professor Marcone, que tanto me ensina e inspira com o amor, a sabedoria e o entusiasmo com que exerce sua profissão.

Acknowledgments

Chegar a esta página desta tese, da minha vida, foi desafiador. A jornada foi longa e exigiu de mim muita dedicação, renúncias e resiliência. Porém, nada disso seria suficiente se não houvesse instituições e pessoas apoiando a realização deste trabalho.

Pela oportunidade que me foi concedida de capacitação profissional, agradeço à UFVJM (Universidade Federal dos Vales do Jequitinhonha e Mucuri) e ao seu ICT (Instituto de Ciência e Tecnologia de Diamantina), onde sou professora, à UFMG (Universidade Federal de Minas Gerais) e ao seu PPGEE (Programa de Pós-Graduação em Engenharia Elétrica), onde estou realizando o doutorado, à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e à FAPEMIG (Fundação de Amparo à Pesquisa do Estado de Minas Gerais).

Agradeço aos meus orientador e coorientador, os professores Lucas e Marcone, pela confiança em mim depositada, pelo formidável empenho em me orientar, por todo conhecimento compartilhado, pela assistência prestada e pela agradável convivência. Desejo que o desenvolvimento desta tese tenha sido gratificante para eles (assim como foi para mim), pois eles trabalharam muito nisso.

Agradeço aos membros da comissão do Exame de Qualificação do meu doutorado, os professores André Luiz Maravilha Silva, Eduardo Gontijo Carrano e Geraldo Robson Mateus, pelas valiosas sugestões e contribuições para a continuidade dos meus estudos.

Agradeço aos responsáveis pelo Operations Research and Complex Systems Lab (ORCS Lab/UFMG) por toda a estrutura disponibilizada para o desenvolvimento desta tese; ao André Maravilha e ao André Batista pelo empenho em resolver os problemas técnicos do ORCS Lab.

Agradeço à Mônica pela amizade e apoio nos momentos, sempre corridos, que eu estava em Belo Horizonte. Agradeço também a todos os meus amigos que tornaram meus dias mais leves e divertidos durante o doutorado.

Por fim, agradeço àqueles que estão comigo a todo momento e são o meu alicerce, Deus e minha família.

Minha mãe Dê, minhas irmãs Mariana e Carla, minhas sobrinhas Camila e Luísa, Daniela e meu cunhado Cláudio não entendem nada de otimização, heurísticas e etc., mas abraçaram o meu doutorado como se fosse deles e fizeram tudo que podiam para me ajudar. Família, muito obrigada pela maravilhosa convivência, amor, apoio, incentivo, conselhos e por tudo mais que me trouxe alegria, conforto e ânimo para seguir em frente!

Amo-te infinitamente!

Obrigada, Deus! O Senhor é o maestro da minha vida e que me concedeu a benção de chegar até aqui na companhia de pessoas tão especiais. Sim, tu és real e maravilhoso, Deus!

"*Sei que os Teus olhos*
Sempre atentos permanecem em mim
E os Teus ouvidos
Estão sensíveis para ouvir meu clamor."
(Deus de Promessas, Composição: Davi Sacer/Ronald Fonseca/Verônica Sacer)

Resumo

Esta tese aborda o problema de programação de veículos e tripulações com múltiplas garagens (MDVCSP). No MDVCSP, lidamos com dois problemas NP-difíceis de forma integrada: o problema de programação de veículos com múltiplas garagens (MDVSP) e o problema de programação de tripulações (CSP). Para solucionar o MDVCSP, definimos simultaneamente a rotina operacional dos veículos e as jornadas de trabalho das tripulações de um sistema de transporte coletivo por ônibus com múltiplas garagens. Dada a dificuldade de resolver instâncias do mundo real do MDVCSP usando métodos matemáticos exatos, propomos um algoritmo matheurístico para resolvê-lo. Este algoritmo matheurístico, nomeado ILS-MDVCSP, combina duas estratégias em uma estrutura baseada em busca local iterada (ILS): um algoritmo branch-and-bound para resolver o MDVSP e um algoritmo baseado no VND (método de descida em vizinhança variável) para tratar os CSPs associados. Comparamos o ILS-MDVCSP proposto com cinco abordagens da literatura que utilizam o mesmo conjunto de instâncias para teste. Também resolvemos um problema real de uma das maiores cidades do Brasil. Para este problema, propusemos uma formulação baseada em uma rede tempo-espaço para resolver o subproblema MDVSP. Os resultados obtidos mostraram a eficácia do ILS-MDVCSP, principalmente para lidar com problemas do mundo real e de grande escala. O algoritmo foi capaz de resolver as maiores instâncias da literatura, para as quais não havia solução relatada. Em relação ao tempo de execução, à medida que o tamanho das instâncias aumenta, nossa abordagem torna-se substancialmente menos onerosa que as demais da literatura. Para as instâncias brasileiras, o ILS-MDVCSP economizou, em média, o uso de 25 veículos por dia e reduziu em média 16% o tempo operacional diário dos veículos considerando quatro garagens juntas.

Palavras-chave: Busca local iterada. Matheurística. Programação de veículos e tripulações com múltiplas garagens. Transporte público. Rede tempo-espaço. Descida em vizinhança variável.

Abstract

This thesis addresses the multiple-depot vehicle and crew scheduling problem (MDVCSP). In MDVCSP, we deal with two NP-hard problems in an integrated way: the multiple-depot vehicle scheduling problem (MDVSP) and the crew scheduling problem (CSP). For solving the MDVCSP, we define the vehicles' operational routine and the workdays of the crews of a public bus transport system with multiple depots simultaneously. Given the difficulty of solving real-world instances of the MDVCSP using exact mathematical methods, we propose a matheuristic algorithm for solving it. This matheuristic algorithm, named ILS-MDVCSP, combines two strategies into an iterated local search (ILS) based framework: a branch-and-bound algorithm for solving the MDVSP and a variable neighborhood descent (VND) based algorithm for treating the associated CSPs. We compared the proposed ILS-MDVCSP with five approaches in the literature that use the same benchmark test instances. We also solved a real-world problem of one of Brazil's largest cities. For this problem, we proposed a formulation based on a time-space network to address the MDVSP subproblem. The results obtained showed the effectiveness of ILS-MDVCSP, mainly to deal with real-world and large-scale problems. The algorithm was able to solve the largest instances from the literature, for which there was no reported solution. Regarding the run time, as the instances' size increases, our approach becomes substantially less costly than the others from the literature. For the Brazilian instances, the ILS-MDVCSP saved, on average, the use of 25 vehicles per day and reduced on average by 16% the daily operational time of the vehicles considering four depots together.

Keywords: Iterated local search. Matheuristic. Multiple-depot vehicle and crew scheduling. Public transportation. Time-space network. Variable neighborhood descent.

List of Figures

1.1 Operational planning of public urban bus transport. 15

2.1 Subproblems in a part of a branch-and-bound tree. 32

3.1 Example of vehicle itinerary and its crew duty. 40
3.2 Time-space network layer for the MDVSP. 43
3.3 Flow decomposition possibilities. 46
3.4 Circulation arcs of the timeline of a depot in a Brazilian real-world problem. . 47
3.5 Definition of one network layer for each depot-period. 49
3.6 Example of a solution s^v for the MDVSP. 52
3.7 Example of a solution s^c for the CSP. 52
3.8 MDVSP moves. 54
3.9 CSP moves. 55

List of Tables

2.1 Summary of the literature review for the MDVCSP 26

3.1 Example of part of a timetable 38
3.2 Example of a deadhead matrix 38
3.3 Concepts involved in defining the MDVCSP 42
3.4 Example of direct partitioning of the vehicle itinerary into pieces of work . . . 63
3.5 Example of inverse partitioning of the vehicle itinerary into pieces of work . . 63

4.1 Costs considered in the evaluation functions and their respective values 70
4.2 Characteristics of the different duty types 71
4.3 Parameters of the proposed algorithm 72
4.4 Results from literature instances 75
4.5 Variability of the solutions obtained by the ILS-MDVCSP algorithm 76
4.6 Characteristics of the Belo Horizonte instances 77
4.7 Results from the Belo Horizonte instances (2 depots) 80
4.8 Results from the Belo Horizonte instances (3 depots) 81
4.9 Results from the Belo Horizonte instances (4 depots) 82
4.10 Improvement of the initial solutions in the ILS-MDVCSP 83
4.11 Characteristics of the exact approach for the VSP instances 83

Contents

1 Introduction **14**

1.1 Motivation . . . 14

1.2 Purpose of the Thesis . . . 18

1.3 Text Organization . . . 18

2 Literature Review **20**

2.1 Related Work . . . 20

2.2 Combinatorial Optimization Techniques . . . 27

2.2.1 Local Search . . . 27

2.2.1.1 Variable Neighborhood Descent . . . 28

2.2.2 Metaheuristics . . . 29

2.2.2.1 Iterated Local Search . . . 30

2.2.3 Branch-and-Bound . . . 31

2.2.4 Matheuristics . . . 32

2.3 Combinatorial Optimization Problems . . . 33

2.3.1 Minimum Cost Flow Problem and Multicommodity Flow Problem . 33

2.3.2 Set Partitioning Problem . . . 35

3 Multiple-Depot Vehicle and Crew Scheduling Problem **37**

3.1 Problem Definition . . . 37

3.2 Modeling Approach . . . 41

3.2.1 A Literature Modeling Approach for the MDVCSP . . . 42

3.2.1.1 Time-Space Network for the MDVSP . . . 43

3.2.1.2 Mathematical Formulations for the MDVSP and MDVCSP 44

3.2.2 Proposed Modeling Approach for the MDVSP and MDVCSP . . . 46

3.2.2.1 Proposed Time-Space Network for the MDVSP . . . 47

3.2.2.2 Proposed Mathematical Formulations for the MDVSP and MDVCSP . . . 49

3.3 Solution Approach . . . 51

3.3.1 Solution Representation . . . 51

3.3.1.1 Solution Representation for the MDVSP . . . 51

3.3.1.2 Solution Representation for the CSP . . . 52

3.3.1.3 Solution Representation for the MDVCSP . . . 52

3.3.2 Neighborhood Structures . . . 53
3.3.2.1 MDVSP Neighborhood Structures . . . 53
3.3.2.2 CSP Neighborhood Structures . . . 55
3.3.3 Evaluating Function . . . 56
3.3.4 Matheuristic Algorithm for the MDVCSP . . . 57
3.3.5 Heuristic Methods for the CSP . . . 59
3.3.5.1 Heuristic Methods for Generating the Pieces of Work . . . 60
3.3.5.2 Heuristic Method for Inserting a Piece of Work Into Crew Schedule . . . 64
3.3.5.3 Heuristic Method for Generating an Initial Solution for the CSP . . . 66
3.3.5.4 Heuristic Method of Local Search for the CSP . . . 66

4 Computational Experiments **69**
4.1 Literature Instances . . . 70
4.1.1 Instances Description . . . 70
4.1.2 Parameter Settings . . . 71
4.1.3 Results . . . 73
4.2 Belo Horizonte Instances . . . 76
4.2.1 Instances Description . . . 76
4.2.2 Results . . . 77

5 Conclusions **84**

Bibliography **86**

Chapter 1

Introduction

1.1 Motivation

The planning process of the public bus transport system is highly complex. Therefore the companies of the sector usually decompose it into several subproblems that are approached in three stages: strategic, tactical, and operational.

Strategic planning addresses problems that involve long-term decisions. This planning aims to meet users' demands and, at the same time, respect established budget limitations. Thus, it specifies the bus transit routes through the city (i.e., the set of line routes).

Tactical planning defines the frequency with which the bus routes must be traveled and the timetable. The timetable is composed of the daily trips to be performed by the public transport company. Each trip has times and locations of the start and end.

Figure 1.1 shows the operational planning problems, namely the vehicle scheduling problem (VSP), the crew scheduling problem (CSP), and the crew rostering problem (CRP).

The VSP consists of determining a daily operating routine for a vehicle fleet. Its objective is to make it possible to execute all timetable trips, reduce costs and, at the same time, respect all operational restrictions. As a result, vehicle itineraries are obtained.

There must be a crew (possibly a driver and a collector) responsible for each vehicle's activity in the fleet. So, the CSP deals with defining the workdays (duties) for the crews. When resolving the CSP, we sought to minimize labor costs; however, obligatorily in compliance with the operational and labor rules in force.

At last, there is the CRP. This problem consists of assigning duties to each crew to define their monthly work routine. Thus, in the CRP, specific rules regarding long periods are considered and therefore were not addressed in the daily schedule.

The division into stages presented above follows Desaulniers and Hickman [2007]; Ibarra-Rojas et al. [2015]. These works provide a comprehensive review of models and approaches to solve the subproblems associated with each stage.

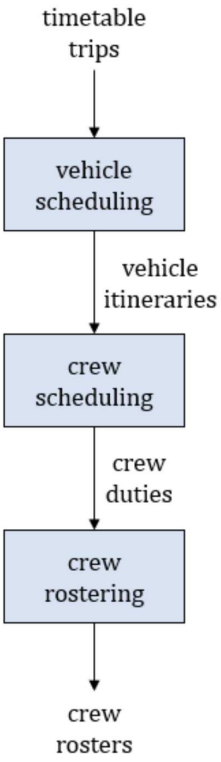


Figure 1.1: Operational planning of public urban bus transport.

For a long time in the literature, proposals for sequential and independent resolution of the subproblems of the planning process of the public bus transport system predominated. However, this type of approach leads to suboptimal solutions for global planning. Thus, with the development of efficient computers and optimization techniques, more and more successful research has emerged to integrate some of these subproblems [Steinzen et al., 2010; Mesquita et al., 2013; Borndörfer et al., 2017; Carosi et al., 2019; Perumal et al., 2021; Er-Rbib et al., 2021; Liang et al., 2020].

In the scope of operational planning, some authors deal with the integrated vehicle and crew scheduling problem (VCSP) [Ball et al., 1983; Patrikalakis and Xerocostas, 1992; Haase et al., 2001; Freling et al., 2003; Laurent and Hao, 2008; Simões et al., 2011]. In the VCSP, vehicle itineraries and the workdays of the crews are defined simultaneously. Solving VSP and CSP in an integrated way is useful, as there is a dependency relationship between these problems. Operating costs can even be an opposite relationship; that is, a characteristic favorable to a solution for the VSP does not always reflect satisfactorily in the CSP and vice versa.

In this thesis, we approach the VCSP with multiple depots. In the so-called multiple-depot vehicle and crew scheduling problem (MDVCSP), each timetable trip may be assigned to a subset of existing depots. In addition, each depot has its vehicles and crews. Therefore, solving more than one depot simultaneously (rather than distributing trips between depots and then solving each depot separately) makes the problem more flexible and more likely to result in better solutions.

The benefits of considering multiple depots were initially identified in solving the multiple-depot vehicle scheduling problem (MDVSP), and there is a vast literature that addresses this problem [Kliewer et al., 2006; Pepin et al., 2008; Desfontaines and Desaulniers, 2018; Kulkarni et al., 2018; Dauer and Prata, 2021].

MDVCSP is an even more complex problem than MDVSP and has also received the attention of researchers [Gaffi and Nonato, 1999; Huisman et al., 2005; Borndörfer et al., 2008; Mesquita and Paias, 2008; Steinzen et al., 2010; Kliewer et al., 2012; Horváth and Kis, 2019]. The works that address the MDVCSP differ significantly in terms of the assumptions considered. Gaffi and Nonato [1999], for example, impose that a crew is assigned to the same vehicle during the whole duty. On the other hand, Mesquita and Paias [2008] consider that a crew can change vehicles at any time at an appropriate local. There are still works that allow the exchange of vehicles, but only between work shifts of the crew [Huisman et al., 2005; Steinzen et al., 2010; Kliewer et al., 2012; Horváth and Kis, 2019]. Thus, the particularities of the works often make it impossible to compare them.

This thesis approaches the MDVCSP following the same assumptions considered in Huisman et al. [2005]; Borndörfer et al. [2008]; Steinzen et al. [2010]; Horváth and Kis [2019]. In addition, we also deal with a Brazilian real-world problem.

We propose a formulation for the MDVSP based on the time-space network elaborated by Steinzen et al. [2010]. In the Brazilian problem that we address, trips are distributed throughout the day on the timetable. Therefore, we must guarantee a minimum daily stay in the depot for each vehicle in our formulation. This time is for the maintenance and cleaning of the vehicles. Steinzen et al. [2010] and other authors [Huisman et al., 2005; Steinzen et al., 2007; Borndörfer et al., 2008; Steinzen et al., 2010; Horváth and Kis, 2019] do not consider this situation. In the problem they address, there is a long interval of the day in which no trip is scheduled on the timetable.

Furthermore, it is possible to combine our formulation for the MDVSP with a set partitioning formulation for the CSP and thus constitute a model for the MDVCSP.

MDVCSP is an NP-hard problem [Huisman et al., 2005; Steinzen et al., 2010]. As shown in Bertossi et al. [1987] for the MDVSP and in Fischetti et al. [1989] for the CSP, each of these problems is NP-hard. Therefore, we approach the MDVCSP through a matheuristic algorithm to address real-world instances, which generally involve a high number of variables.

Our matheuristic algorithm, named ILS-MDVCSP, uses the iterated local search (ILS) [Stützle and Ruiz, 2018] framework for exploring the solution space of the MDVCSP. In turn, its solution process combines two methods: a branch-and-bound method [Wolsey, 2020] to solve the MDVSP in optimality and a heuristic procedure based on the variable neighborhood descent (VND) [Hansen et al., 2017] to treat associated CSPs.

To the best of our knowledge, only Steinzen et al. [2007] addressed the same prob-

lem using a matheuristic. They deal with the MDVCSP using a hybrid evolutionary algorithm (EA) to assign trips to depots. In this context, an individual is a trip-depot vector where each trip is assigned to a single depot. In addition, these authors used Lagrangian heuristics based on column generation to define vehicle and crew schedules and thus evaluate an individual's fitness. The results showed that the hybrid EA overcame a sequential approach to solving the MDVSP and CSP. However, the hybrid EA achieved lower performance than the integrated resolution algorithms proposed in Borndörfer et al. [2004]; Huisman et al. [2005]. As well as Steinzen et al. [2007], we compared the ILS-MDVCSP against the approaches proposed in Huisman et al. [2005] and Borndörfer et al. [2008] (the improved work of Borndörfer et al. [2004]). Unlike Steinzen et al. [2007], our algorithm performed better for all instances, considering the results reported in these works.

To validate our algorithm, we perform experiments with benchmark test instances and compare the results against those reported in other well-known approaches from the literature. We also present and discuss the results obtained for a real-world problem of a city in Brazil, where the bus is the most popular form of public transport in large urban centers. We compare the companies' solutions (VSP and CSP sequential planning) with the solutions obtained from two integrated approaches: VCSP resolution and MDVCSP resolution. The results obtained showed:

1. The effectiveness of the matheuristic algorithm ILS-MDVCSP, mainly to deal with real-world and large-scale problems.

2. The integrated resolution of the VSP and CSP is more efficient than the traditional sequential resolution of these problems. Furthermore, the integrated approach with multiple depots is more effective than the one with a single depot.

Finally, it is worth noting that, together with population growth and the new challenges of urban mobility, the demand for an efficient, safe, quality, and cost-effective locomotion service has been increasing. Concurrently, alternatives to move in the cities have emerged. Therefore, the urban bus sector needs to adapt to this new competitive and demanding scenario. In this respect, the optimized definition of vehicle and crew schedules, as we propose in this thesis, is essential. It is directly related to the working conditions of the employed labor and involves most of the sector's operating costs. In Brazil, according to the NTU (National Association of Urban Transport Companies), half of the cost of the transport system refers to labor costs and almost 27% to fuel.

1.2 Purpose of the Thesis

We hope to achieve the following purposes with this thesis:

1. To propose models and techniques to solve, in an integrated way and considering multiple depots simultaneously, the following problems of the operational planning of the public bus transport system: vehicle scheduling and crew scheduling.
2. To propose a time-space network for the MDVSP that ensures the definition of valid vehicle operating periods. It is necessary when there is no on a 24-hour planning horizon a sufficient interval in which no trip is scheduled. That is, we define all the intervals of the day (operating periods) in which vehicles can work without extrapolating the maximum daily operating time. The objective is to impose a minimum time in the depot for each vehicle. This time is necessary and can be used for cleaning, inspecting, maintaining, and refueling the vehicle.
3. Show the efficiency of the proposed approach to solve instances widely used in the literature.
4. Show the applicability and effectiveness of the proposed approach to deal with real-world and large-scale problems.
5. To present an alternative for the Brazilian public bus transport system to deal with the growth in expenses. Optimized planning of the sector can avoid increased fares and a government subsidy while favoring the sector's efficiency, competitiveness, and profitability.
6. Publish the scientific knowledge produced and, therefore, support research related to the problem and the combinatorial optimization techniques we address. From this thesis, we generated the article described below.

   Simões, E. M. L.; Batista, L. S.; Souza, M. J. F.. A Matheuristic Algorithm for the Multiple-Depot Vehicle and Crew Scheduling Problem. IEEE Access, v. 9, p. 155897-155923, 2021.

1.3 Text Organization

The remaining of this thesis is as follows.

Chapter 2 reviews some related works to the MDVCSP (Section 2.1) and provides the necessary background for the algorithm and models proposed for this problem (Sections 2.2 and 2.3, respectively).

Chapter 3 is composed of the sections described next. Section 3.1 defines the MDVCSP and details the constraints we consider to solve a real-world problem of a city in Brazil. Section 3.2 describes the literature formulations for the MDVSP and MDVCSP, which support our formulation. Section 3.2 also presents the formulation we propose to deal with a Brazilian real-world problem. These formulations are based on the time-space network representations of the MDVSP detailed in Subsections 3.2.1.1 and 3.2.2.1. The problem-solving approach is detailed in Section 3.3. In this section, Subsection 3.3.1 describes how to represent the MDVSP, CSP, and MDVCSP. Subsection 3.3.2 presents the neighborhood structures used to explore the solution space of these problems. Subsection 3.3.3 defines the evaluation functions to vehicle and crew schedules. Subsection 3.3.4 presents the proposed ILS-MDVCSP matheuristic algorithm for solving the MDVCSP. Subsection 3.3.5 describes heuristic procedures to treat the CSP exclusively and which are auxiliary methods for the ILS-MDVCSP.

Chapter 4 presents and discusses the results obtained for benchmark test instances (Subsection 4.1) and Brazilian instances (Subsection 4.2).

Finally, concluding remarks are pointed out in Chapter 5.

Chapter 2

Literature Review

This chapter reviews state-of-the-art approaches for the MDVCSP in Section 2.1 and some techniques and problems of Operations Research used in this thesis (Sections 2.2 and 2.3).

Section 2.2 concerns combinatorial optimization techniques that are the foundation of the proposed matheuristic algorithm for the MDVCSP and is organized as follows: local search - variable neighborhood descent (Subsection 2.2.1), metaheuristic - iterated local search (Subsection 2.2.2), Branch-and-Bound (Subsection 2.2.3), and matheuristics (Subsection 2.2.4).

Section 2.3 reviews Minimum Cost Flow Problem and Multicommodity Flow Problem in Subsection 2.3.1 and Set Partitioning Problem in Subsection 2.3.2. These problems are the base of the mathematical formulations for the MDVCSP that we will discuss.

2.1 Related Work

VSP and CSP have long been studied extensively in the Operations Research (OR) literature. Among the precursors are the works Elias [1964]; Kirkman [1968]; Saha [1970]; Wren [1972]; Booler [1975]; Wren [1981]. In the 1980s, Ball et al. [1983] already pointed out the advantages of addressing these problems in an integrated manner and proposed a heuristic resolution method for the VCSP.

Patrikalakis and Xerocostas [1992] proposed the first mathematical formulation for the VCSP. However, this model was for illustrative purposes only, being computationally intractable [Freling et al., 1999]. Thus, the authors in Freling et al. [1995] are considered the pioneers in proposing an integer programming formulation for the problem. This work gave rise to a series of studies developed by Freling *et al.* on the topic, such as Freling [1997]; Freling et al. [1999, 2003]. The formulations presented in these works are similar and consist basically of two parts:

1. A quasi-assignment formulation based on a $G(V, A)$ network ensures the viability of the VSP. In this network, V is the set of all timetable trips and two more nodes,

s (source) and t (sink), both representing the depot. The set A corresponds to the arcs that: a) connect s and t to all other nodes in V, and b) connect compatible trips, i.e., trips that can be performed by the same vehicle. Thus, a feasible solution for the VSP is a set of disjoint paths from s to t in the network G, in such a way that each trip node in V is covered precisely once;

2. A set partitioning formulation to assign each vehicle activity to a crew.

With the transformations in urban public transportation resulting from population growth, social changes, and policies to encourage sustainability, recent research has addressed more complex variants of the VCSP. Perumal et al. [2021] solved the VCSP with electric buses (E-VCSP). So, new restrictions are imposed, such as i) limited driving range of electric vehicles, ii) longer recharging times of vehicles, and iii) fixed charging locations. Boyer et al. [2018] and Andrade-Michel et al. [2021] addressed the VCSP by treating each vehicle and crew individually. They considered the compatibility between each pair of vehicle-driver (the driver has/hasn't the ability to drive the vehicle model), driver-line (driver knows or not the line route), and vehicle-line (the bus must have appropriate characteristics to meet the line). Thus, the VCSP solution is a feasible trip-vehicle-driver assignment for each line. Furthermore, to resolve the VCSP, Andrade-Michel et al. [2021] considered the reliability of each driver (probability that he will not be absent from work) and the importance of each trip (measured by the number of passengers it serves). Amberg et al. [2019] observed that interruption (delay) can occur in urban public transport due to heavy traffic or passenger behavior. In addition, this initial delay of a scheduled task can cause delays in other activities that use the same resources (vehicle and crew). Thus, Amberg et al. [2019] solved the VCSP aiming to minimize operational costs and, at the same time, define robust vehicle and crew schedules. The robustness in this context concerns the ability of an integrated schedule to prevent the spread of delays. These previous works and many others solved the VCSP considering that there was only one depot for managing vehicles and crews (see e.g., Laurent and Hao [2008]; Simões et al. [2011]; Kang et al. [2019]). This problem is so-called single-depot VCSP (SDVCSP).

Medium-sized and large public transport companies usually have multiple depots to manage their vehicles and crews. In this scenario, we have the definition of the multiple-depot VCSP (MDVCSP).

Based on Freling et al. [2003], Huisman et al. [2005] proposed a model and algorithm to deal with the MDVCSP. The developed algorithm combined column generation with Lagrangian relaxation and consisted of the following steps:

1. Generation of the initial columns: Solved the problems MDVSP and several CSPs to generate the initial columns, i.e., the crew duties. To solve the MDVSP, the authors used the model described in Huisman et al. [2004] and the all-purpose solver CPLEX. To address the CSP, they considered the approach presented in Freling et al. [2003];

2. Obtaining the lower bound: Used column generation with Lagrangian relaxation. The problems addressed are:

   - Master problem: By relaxing the necessary constraints, the master problem becomes a single and large VSP with a single depot, which can be solved in polynomial time;
   - Pricing problem: This problem was solved in two phases, considering one depot at a time, to generate new columns (crew duties):

     a) Creating pieces of work: Each piece of work was a sequence of trips of a vehicle limited by a minimum and maximum duration. In this phase, Huisman et al. [2005] used a piece-generation network. In this network, each node corresponded to a relief point defined by location and time, where and when the change of crews might occur; each arc had an associated reduced cost and corresponded to a trip or a deadhead (vehicle travel without passengers). Then, the authors solved a shortest path problem between each pair of nodes in the network to create pieces of work;

     b) Generation of new duties: The work proposed a simplification and considered i) the formation of duties with only two pieces of work and ii) the reduced cost of duty was the sum of the reduced costs of the pieces of work that composed it. The latter simplification was possible because of the continuous attendance assumption, i.e., whenever a vehicle is out of the depot, there is a crew with it.

3. Obtaining a feasible solution: Used Lagrangian relaxation to solved the MDVSP from the Lagrangian multipliers from the previous stage. Thus, good feasible solutions for MDVSP were obtained in few iterations. Finally, from the MDVSP solution, several CSPs, one for each depot, were addressed.

Huisman et al. [2005] developed another model for the MDVCSP from the formulation proposed by Haase et al. [2001] for the VCSP. However, their algorithm generated better results using the model based on Freling et al. [2003].

Before Huisman et al. [2005], only Gaffi and Nonato [1999] addressed the MDVCSP. Gaffi and Nonato [1999] used a model and algorithm similar to those presented in Huisman et al. [2005], but they consider particular constraints that facilitate the resolution of the problem [Huisman et al., 2005].

Based on Freling [1997]; Freling et al. [1999, 2003] and using one of the models proposed in Huisman et al. [2005], Borndörfer et al. [2008] proposed a Lagrangian relaxation and column generation approach to the MDVCSP. For the solution of the Lagrangian relaxations, they used a proximal bundle method. Besides, they used the primal and dual

information generated by this bundle method to guide a branch-and-bound algorithm to produce integer solutions.

Mesquita and Paias [2008] proposed two formulations for the MDVCSP. The first (SP-VCSP) was similar to the one presented in Huisman et al. [2005] but with a smaller number of constraints and decision variables. This formulation combined two models: the multicommodity network flow model for vehicle scheduling and the set partitioning/covering model for crew scheduling. The second formulation (SPC-VCSP) was an extension of the previous one, and it replaced some set partitioning constraints by set covering constraints. This modification made the model more flexible. Without increasing the complexity of the problem, it made possible situations in which the crew was allowed to change vehicles during their duty. Unlike Huisman et al. [2005] and other authors (e.g. Borndörfer et al. [2008]; Steinzen et al. [2010]; Horváth and Kis [2019]), Mesquita and Paias [2008] allowed a given crew to drive vehicles from any depot and not just the depot to which they belong. Furthermore, a crew could change from a vehicle to another at any time at an appropriate local. Therefore, the results in Mesquita and Paias [2008] cannot be directly compared with those of other studies that addressed the MDVCSP.

Steinzen et al. [2010] presented a new formulation for the MDVCSP, based on the so-called time-space network. Until then, in the literature, all works used models similar to the one proposed in Huisman et al. [2005], based on the so-called connection-based network to address the MDVCSP. The network proposed in Steinzen et al. [2010] was much smaller considering the number of nodes and arcs, which resulted in a mathematical formulation with a much smaller number of constraints and variables. The proposed solution methodology was similar to that presented in Huisman et al. [2005] and combined the generation of columns with Lagrangian relaxation. It has two major phases:

1. Lagrangian relaxation phase: It aimed to determine a good set of columns (crew duties) and a lower bound for the relaxed problem. Therefore, the restricted master problem considered was the original relaxed problem using Lagrangian relaxation, and Steinzen et al. [2010] solved it approximately with the subgradient method. In the pricing problem, they obtained new columns with a negative reduced cost of a time-space network. Therefore, they used a resource-constrained shortest-path (RCSP) formulation. In this approach, any path from the source node to the sink node in the time-space network corresponded to a feasible duty. The path's cost corresponded to the reduced cost of the duty, and the authors considered resources to satisfy the constraints associated with a feasible duty. Based on the proposed formulation, Steinzen et al. [2010] used a dynamic programming algorithm to obtain negative reduced costs duties efficiently;

2. Integer programming phase: It aimed to determine a feasible integer solution. In this phase, Steinzen et al. [2010] proposed a Lagrangian relaxation approach in a time-

space network. Two improvements were essential for the computational performance gain: a branch-and-price heuristic algorithm and a variable fixing heuristic.

For the tests, Steinzen et al. [2010] used the instances available at Huisman [2003] and generated larger ones using the same algorithm of Huisman et al. [2005], which are available in Steinzen [2007a]. According to Steinzen et al. [2010], when considering other works in the literature, their results were the best in terms of processing time (when it was possible to compare) and quality of the solution generated (number of vehicles and crews employed).

As we mentioned earlier, Steinzen et al. [2007] addressed the MDVCSP using a hybrid evolutionary algorithm (EA). In the proposed strategy, the hybrid EA was used to assign trips to depots and, in this way, transform the MDVCSP into several VCSPs. Both problems, MDVCSP and VCSP, are NP-hard. However, in VCSP, the associated vehicle scheduling problem can be solved in polynomial time, as it is the minimum cost flow problem. Thus, the hybrid EA used Lagrangian heuristics based on column generation to solve the VCSP and compute the individuals' fitness.

The solution approaches described in Steinzen et al. [2007] and Steinzen et al. [2010] provided the basis for Kliewer et al. [2012] to solve the MDVCSP with time windows for scheduled trips (MDVCSP-TW). At MDVCSP-TW, the scheduled time for timetable trips is not fixed. That is, at MDVCSP-TW, the trip departure and arrival times are variables. This flexibility increases the number of trips compatible with each other and can reduce required vehicles and crews in the schedule. Kliewer et al. [2012] compared the approaches a) traditional sequential planning of vehicles and crews, b) sequential planning extended with consideration of time windows in the vehicle scheduling phase, c) integrated planning, and d) integrated planning with time windows. The results indicated that the integrated planning approach with time windows was better than the others.

Ciancio et al. [2018] solved the MDVCSP considering several real-world restrictions according to the European Union legal framework. These restrictions are so specific that their proposed problem is quite different from those found in the literature [Ciancio et al., 2018]. The authors proposed an ILS-based heuristic to address the MDVSP and a greedy heuristic combined with local searches to build an initial solution for the CSP. In integration, these solutions are modified by changing the trips' allocation on vehicles to minimize the combined objective function. The authors used instances proposed by themselves in the tests. The largest instance considered had 712 trips and 4 depots.

Horváth and Kis [2019] proposed a mathematical formulation for the MDVCSP based on the model of Steinzen et al. [2010]. Furthermore, they presented a branch-and-price procedure to approach the problem exactly. This approach used an efficient pricing method, some branching strategies, and a simple primal heuristic. According to the authors, this was the first work to propose an exact solution to the MDVCSP defined

in Huisman et al. [2005]. The proposed approach was able to efficiently solve only small instances, with 4 depots and 80 or 100 trips. Optimal solutions were found for 4 of the 20 instances considered and, for the others, the GAP of the solution found was defined concerning the lower bound obtained. The GAP was less than 0.5% for 7 instances.

The formulations proposed in the literature for the MDVCSP represent the associated MDVSP in two ways: 1– connection-based network, initially proposed by Huisman et al. [2005]; or 2– time-space network [Steinzen et al., 2010]. As shown in Kliewer et al. [2006] and Steinzen et al. [2010], in the time-space network the amount of deadhead arcs is much less than in the connection-based network. Kliewer et al. [2006] solved very large practical instances of the MDVSP in optimality through the direct application of standard optimization software. Thus, we propose a formulation based on a time-space network for the MDVSP subproblem in this thesis. Our time-space network for the MDVSP ensures the definition of valid vehicle operating periods. It is necessary when there is no on a 24-hour planning horizon a sufficient interval in which no trip is scheduled. The revised literature does not consider this situation because the problems addressed have a long interval of the day without a trip.

Table 2.1 summarizes the approaches proposed to the MDVCSP from the literature review and present work. This table also describes the instances considered in the works' computational experiments.

Table 2.1 shows that the literature applies different optimization techniques to tackle the MDVCSP (mathematical programming, evolutionary algorithms, metaheuristics, and matheuristics). Most research proposed a solution approach that combines the generation of columns with Lagrangian relaxation. Our work proposes a matheuristic that combines branch-and-bound and variable neighborhood descent into an iterated local search-based framework to approach the MDVCSP. To the best of our knowledge, we are the first to address the largest instances from the benchmark instances widely used in the literature. Furthermore, we present the largest real-world instance (1573 trips and four depots) and solve it.

Regarding the run time, as the size of the instances increases, our approach becomes substantially less costly than the others from the literature. Thus, we show that the proposed algorithm can efficiently handle large-scale instances from literature and the real-world.

In the remainder of this chapter, we provide the necessary background for the algorithm and models used in this thesis. These are comprehensive subjects. Thus, we consider only a few topics directly related to our solution approach.

Table 2.1: Summary of the literature review for the MDVCSP

Reference	Solution approach		Literature instances		Real-world instances	
	Type	Approach	Generated by	Largest instance treated	Companies from	Largest instance treated
Gaffi and Nonato [1999]	Heuristic	CG-LR[1]	-	-	Italy[9]	257 trips, 28 depots
Huisman et al. [2005]	Heuristic	CG-LR	themselves	200 trips, 4 depots	Netherlands[10]	653 trips, 1.74 depot/trip[12]
Steinzen et al. [2007]	Heuristic	EA-CG-LR[2]	Huisman et al. [2005]	200 trips, 4 depots	-	-
Borndörfer et al. [2008]	Heuristic	CG-LR	Huisman et al. [2005]	400 trips, 4 depots	Germany[10]	1414 trips, 3 depots
Mesquita and Paias [2008]	Heuristic	CG-LPR-BB[3]	Huisman et al. [2005][7]	400 trips, 4 depots	-	-
Steinzen et al. [2010]	Heuristic	CG-LR	Huisman et al. [2005][8]	640 trips, 4 depots	-	-
Kliewer et al. [2012]	Heuristic	CG-LR	Huisman et al. [2005][8]	640 trips, 4 depots	-	-
Ciancio et al. [2018]	Heuristic	ILS-GH-LS[4]	themselves	712 trips, 4 depots	-	-
Horváth and Kis [2019]	Exact	BP[5]	Huisman et al. [2005]	100 trips, 4 depots	-	-
Present work	Heuristic	ILS-BB-VND[6]	Huisman et al. [2005][8]	800 trips, 4 depots	Brazil[11]	1573 trips, 4 depots

[1] Column generation in combination with Lagrangian relaxation.
[2] Hybrid evolutionary algorithm and column generation in combination with Lagrangian relaxation.
[3] Column generation in combination with linear programming relaxation; branch-and-bound.
[4] ILS-based heuristic and a greedy heuristic combined with local searches.
[5] Branch-and-price.
[6] Combines branch-and-bound and variable neighborhood descent into an iterated local search-based framework.
[7] Unlike other authors, Mesquita and Paias [2008] allowed a given crew to drive vehicles from any depot.
[8] Includes the instances generated by Huisman et al. [2005] and Steinzen et al. [2010]. Both used the instances generation algorithm of Huisman et al. [2005].
[9] A driver must be assigned a single vehicle during the whole duty in this problem. Furthermore, there are different types of vehicles, and for each trip the suggested type is known, corresponding to the most suitable type for that service.
[10] Some trips have to be assigned to vehicles and drivers from a certain subset of depots in this problem.
[11] Every trip can be assigned to any depot in this problem.
[12] The average number of depots to which a trip may be assigned.

2.2 Combinatorial Optimization Techniques

Combinatorial optimization techniques aim to obtain the best solution to a problem that has a finite but extremely large set of feasible solutions. The best solution is called the optimal solution (global optimum), and it is not necessarily unique. That is, there can be several optimal solutions to the same problem. The quality of a solution s, from the set of feasible solutions S, is defined by a function $f : S \to \mathbb{R}$, called objective function or evaluation function. This function associates to each solution $s \in S$ a real-valued $f(s)$. When the objective is to obtain a solution s^* with the smallest possible value for f, we have a minimization problem. Otherwise, that is, when the objective is to find a solution s^* with the highest possible value for f, we have a maximization problem.

Many practical instances of combinatorial optimization problems cannot be solved exactly in a reasonable time. In these cases, heuristics should be used to produce an approximate solution of high quality quickly, or sometimes an optimal solution but without proof of its optimality. Many of these heuristics employ neighborhood structures to explore the solution space. Let $\min\{f(s) \mid s \in S\}$ be a minimization optimization problem, let $\mathcal{N}(s)$ denote a neighborhood structure of a given solution s. $\mathcal{N}(s)$ usually consists of all solutions obtained from s by some simple modification, i.e., $\mathcal{N} : S \to \mathcal{P}(S)$, where $\mathcal{P}(S)$ denotes the power set of the set S. Then, a solution s^* is a local optimum, relative to neighborhood $\mathcal{N}(s^*)$, if $f(s^*) \leq f(s), \forall s \in \mathcal{N}(s^*)$. Furthermore, as previously described, a solution s^* is an optimal solution if $f(s^*) \leq f(s), \forall s \in S$.

Vehicle scheduling, crew scheduling, and related problems are classical combinatorial optimization problems. Therefore, this section briefly explains the combinatorial optimization techniques we applied to address the MDVCSP.

2.2.1 Local Search

Local search is a fundamental tool in developing heuristic algorithms and explores a neighborhood structure of a current solution at each iteration [Hansen et al., 2017; Amaral et al., 2021]. From the neighborhood of the current solution, a local search heuristic chooses one feasible solution that improves the value of the objective function. The selected solution becomes the new current solution. This process repeats until the current solution is the local optimum for the neighborhood structure considered.

The most common search strategies used within a local search heuristic are:

1. The *best improvement* that selects at each iteration the neighbor with the best objective function value;

2. The *first improvement* that selects at each iteration the first evaluated neighbor that is better than the current solution. Furthermore, the first improvement that evaluates the neighborhood randomly is equivalent to a random selection of an improving neighbor.

Algorithm 1 describes the local search procedure using the best improvement search strategy for a minimization problem.

Algorithm 1: Local search using the best improvement search strategy

Data: $\mathcal{N}$, operator that defines the neighborhood structure.
Data: s_0, the initial solution.
Result: s, the solution after local search.
$s \leftarrow s_0$
$s' \leftarrow \text{argmin}_{s'' \in \mathcal{N}(s)} f(s'')$
while $f(s') < f(s)$ **do**
    $s \leftarrow s'$
    $s' \leftarrow \text{argmin}_{s'' \in \mathcal{N}(s)} f(s'')$

2.2.1.1 Variable Neighborhood Descent

The *variable neighborhood descent* (VND) is a local search procedure [Hansen et al., 2017, 2019]. This method is based on the principle that a local optimal concerning a given neighborhood structure does not necessarily correspond to a local optimum concerning another neighborhood structure. Thus, the objective is to explore the solution space through systematic changes of neighborhood structures.

There are different VND variants in the literature. Each variant has its own rule for selecting the next neighborhood structure to be explored if the current solution is improved. Hansen et al. [2017] describe four sequential VND variants: basic VND (B-VND), pipe VND (P-VND), cyclic VND (C-VND) and Union VND (U-VND).

We apply a B-VND-based local search for improving crew scheduling in this thesis. Let $N = \{\mathcal{N}_1, ..., \mathcal{N}_{k_{max}}\}$ be a set of operators defining the neighborhood structures and the order of their examination. The *basic sequential VND* (B-VND) procedure [Hansen et al., 2017] iteratively explores its neighborhood structures one after another according to

the established order. As soon as an improvement of the current solution in some neighborhood structure occurs, the B-VND resumes search in the first neighborhood structure of the new current solution. Algorithm 2 details this procedure for a minimization problem and uses the best improvement search strategy.

Algorithm 2: B-VND using the best improvement search strategy

```
1  Data: N = {N_1, ..., N_{k_max}}, the set of operators defining the neighborhood
         structures and the order of their examination.
2  Data: s_0, the initial solution.
3  Result: s, the solution after local search.
4  k ← 1 // neighborhood structure to explore
5  s ← s_0
6  while k ≤ k_max do
7      s' ← argmin_{s''∈N_k(s)} f(s'')
8      if f(s') < f(s) then
9          s ← s'
10         k ← 1
11     else
12         k ← k + 1
```

2.2.2 Metaheuristics

Metaheuristics are generic or higher-level heuristics that aim to explore the problem's solution space intelligently. They have mechanisms so that the search is not stuck in local optima and thus finds good solutions in other regions at an acceptable computational cost.

Metaheuristic algorithms effectively solve a wide variety of optimization problems. In Hussain et al. [2019], there is a comprehensive survey of metaheuristic research in literature and Sörensen et al. [2018] describe the history of metaheuristics in different periods. Osaba et al. [2021] proposed an end-to-end methodology for addressing real-world optimization problems with metaheuristic algorithms.

Next, we present the main ideas of the *Iterated Local Search* metaheuristic. Our algorithm to tackle the MDVCSP uses an *Iterated Local Search* based framework.

2.2.2.1 Iterated Local Search

The *iterated local search* (ILS) metaheuristic [Stützle and Ruiz, 2018; Lourenço et al., 2019] combines perturbation and local search in an iterative process to generate a chain of good solutions. The perturbation modifies the current solution and generally consists of the random move in a neighborhood of higher order than the one used by the local search algorithm. Therefore, the local search should not be able to undo the perturbation.

At each iteration, the ILS performs a perturbation to the current solution and, subsequently, executes a local search procedure. Then, the acceptance criterion of the ILS determines whether the resulting solution is accepted or not as the new current solution. These steps are repeated until the stop condition is satisfied.

Algorithm 3 describes the basic ILS and has four components:

1. *GenerateInitialSolution*: this procedure generates an initial s_0 solution, i.e., the starting point of the search;
2. *LocalSearch*: it is the improvement method that returns an improved solution s after generating the initial solution and another *s"* after perturbation;
3. *Perturbation*: this procedure modifies the current solution s leading to an intermediate solution *s'*. An efficient perturbation is one that does not drastically change the current solution, but still allows the local search to explore different regions on space of all candidate solutions;
4. *AcceptanceCriterion*: it determines whether the resulting solution *s"* is accepted or not as the new current solution.

Algorithm 3: ILS

```
1 Result: s, the best solution found.
2 s_0 ← GenerateInitialSolution()
3 s ← LocalSearch(s_0) // optional line of code
4 while the stop condition is not satisfied do
5     s' ← Perturbation(s, history)
6     s'' ← LocalSearch(s')
7     s ← AcceptanceCriterion(s, s'', history)
```

Algorithm 3 considers aspects of the search *history* in the *Perturbation* and *AcceptanceCriterion* procedures. This history can be used by ILS to, for example, adjust the

perturbation strength or the choice done in the acceptance criterion. Studies show that incorporating memory enhances performance [Lourenço et al., 2010].

According to Stützle and Ruiz [2018], the ILS has advantages compared to repeatedly starting the local search method from random initial candidate solutions. The ILS algorithm generally generates more local optima and local optima of better average quality with a given run time.

The basic ILS is a simple and easy to implement metaheuristic. Furthermore, a high-performing ILS algorithm can be developed when problem-specific details are taken into account [Stützle and Ruiz, 2018; Lourenço et al., 2019].

2.2.3 Branch-and-Bound

Branch-and-bound is an exact solution approach for combinatorial optimization problems that implicitly enumerate many solutions. Land and Doig [1960] proposed this algorithm initially for solving integer programming (IP) problems. Based on the divide and conquer principle, the branch-and-bound decomposes the original problem into smaller subproblems that are easier to solve. Then, the solutions to the subproblems are used to solve the original problem. The decomposition works based on the following proposition [Wolsey, 2020]: let the problem be $Z = \max\{cx : x \in S\}$, let $S = S_1 \cup ... \cup S_K$ be a decomposition of S into smaller sets, and let $Z^k = \max\{cx : x \in S_k\}$ for $k = 1, ..., K$. Then, $Z = \max_k Z^k$.

Figure 2.1 shows a part of a branch-and-bound tree, a typical way to represent the branch-and-bound strategy. In Figure 2.1, each subproblem represents a node on the tree. The main problem S is at the root node, and is then divided into two subproblems, S_1 and S_2 ($S = S_1 \cup S_2$). This process of dividing a problem into smaller subproblems is called *branching* and is performed successively on the subproblems. However, the complete enumeration of the solution space is very time-consuming for most problems of practical size. Therefore, in the branch-and-bound, whenever possible, a node is *pruned*. That is, the node subproblem goes not divided anymore. Branch-and-bound uses lower and upper *bounds* information to prune nodes that cannot contain a solution better than the best solution (*incumbent solution*) found so far. In general, a node is pruned if one of the following cases happens:

1. *Pruning by optimality:* the optimal solution to a node subproblem is found;
2. *Pruning by bound:* the upper bound calculated for a node subproblem is not greater than the lower bound already known (in a maximization problem);

3. *Pruning by infeasibility:* the feasible region of a node subproblem is empty.

Suppose a node cannot be pruned based on the above conditions. In that case, new branches are created to decompose the node subproblem into smaller subproblems. The algorithm stops when all nodes are pruned and the optimal solution will be the *incumbent solution.*

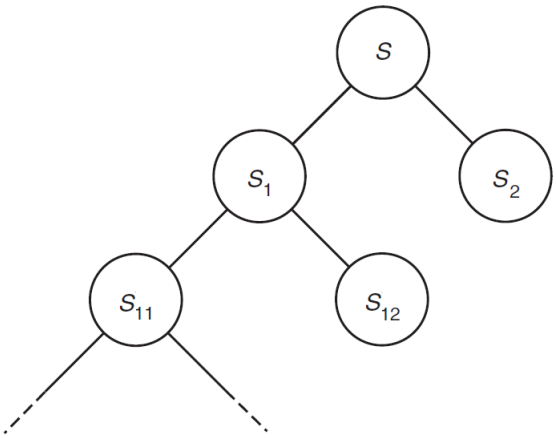


Figure 2.1: Subproblems in a part of a branch-and-bound tree.

The optimization solvers, like Gurobi Optimization, LLC [2021], generally use a linear programming-based branch-and-bound algorithm to solve *mixed-integer linear programming* (MILP) problems. In LP-based branch-and-bound, linear programming relaxations provide the bounds. The capabilities of solvers increased so much in recent years, but the underlying theory has changed relatively little. The main techniques improved have been preprocessing/presolving, cutting planes, primal heuristics, and parallelism.

In this thesis, we use the Gurobi to solve the MDVSP in optimality.

2.2.4 Matheuristics

We propose a *matheuristic* algorithm for solving the MDVCSP. Matheuristics are hybrid algorithms that combine mathematical programming and metaheuristic techniques. According to Raidl [2015], this class of algorithms exploits the individual advantages of these methods and benefits from the synergy between them.

Given the success of the matheuristics, researches in literature have applied this technique to solving several combinatorial optimization problems. Dumitrescu and Stützle [2003] describe approaches in which a local search method is applied to the problem to solve and an exact algorithm to some subproblems. They show that this combination generated powerful algorithms. Puchinger and Raidl [2005] present a more general

classification of existing approaches combining exact and metaheuristic algorithms. They distinguish two main combinations categories: collaborative (the algorithms exchange information, but are not part of each other) and integrative (one technique is a subordinate embedded component of another technique). Jourdan et al. [2009]; Blum et al. [2011]; Raidl [2015]; Talbi [2016] are other reviews of how hybrid approaches are being designed in the literature.

Maniezzo et al. [2021] is a comprehensive tutorial on matheuristics. The book provides a detailed presentation of the best-known metaheuristics and their mathematical hybrids. Furthermore, it describes both extensions of well-known heuristics and innovative, matheuristic-only approaches [Maniezzo et al., 2021].

2.3 Combinatorial Optimization Problems

The field of combinatorial optimization deals with problems of minimizing or maximizing a function of discrete decision variables subject to equality or inequality constraints. This section reviews some well-known combinatorial optimization problems which we use in this thesis to model the MDVCSP.

2.3.1 Minimum Cost Flow Problem and Multicommodity Flow Problem

Network optimization is an important problem domain in operations research. There are a variety of applications in the literature that model practical situations as network optimization problems. See, for example, Mahey et al. [2017].

A lot of well-known network flow problems are special cases of the *minimum cost flow problem*. According to Ahuja et al. [1993], this problem aims to determine a minimum cost shipment of a commodity through a network that will satisfy the flow demands at certain nodes from available supplies at other nodes.

Let $G = (N, A)$ be a directed graph, where N is the set of nodes and A is the set of directed arcs. Each arc $(i, j) \in A$ has an associated cost c_{ij} per unit flow on that arc. Then, we assume that the flow cost varies linearly with the amount of flow. Furthermore, let l_{ij} and u_{ij} be, respectively, the minimum and maximum amount of flow on each arc

$(i, j) \in A$. Each node $i \in N$ has an associated integer b_i which indicates its supply ($b_i > 0$) or demand ($b_i < 0$). If $b_i = 0$, node i is a transshipment node. Let y_{ij} be the decision variables and represents the flow on an arc $(i, j) \in A$. The formulation for the minimum cost flow problem is presented below (2.1)–(2.3).

$$\min \sum_{(i,j)\in A} c_{ij} y_{ij} \tag{2.1}$$

s. t.

$$\sum_{\{j:(i,j)\in A\}} y_{ij} - \sum_{\{j:(j,i)\in A\}} y_{ji} = b_i \quad \forall i \in N, \tag{2.2}$$

$$l_{ij} \leq y_{ij} \leq u_{ij}, \quad \forall (i, j) \in A. \tag{2.3}$$

The objective is to minimize the cost of the flow of a commodity through a network (2.1). Constraints (2.2) ensure that the net flow out of each node (outflow minus inflow) must equal the supply/demand of the node. Constraints (2.3) guarantee that the flow on each arc satisfy the capacity, lower and upper bound, of that arc. We refer to the constraints in (2.2) as *flow conservation constraints* and constraints (2.3) as *flow bound constraints*.

In that model, there are two important details:

1. The data must satisfy the feasibility condition, that is, $\sum_{i\in N} b_i = 0$;
2. The constraint matrix is *totally unimodular*. Then, suppose the supplies/demands of the nodes and the capacities of the arcs are integral. In that case, each solution to the linear program above is integral [Wolsey, 2020].

In general, the minimum cost flow problem is solved by specialized algorithms that run in polynomial time and exploit the structure of the underlying network (see Ahuja et al. [1993, 2001]). These algorithms are often faster than general-purpose linear programming algorithms such as primal or dual simplex.

The *multicommodity flow problem* is an extension of the minimum cost flow problem. In the multicommodity flow problem, several commodities share the same network. Furthermore, each commodity has separate flow conservation constraints, and all commodities share the same flow bound constraints.

Below we give the formulation of the multicommodity flow problem with K as the set of commodities and with separate flow variables and costs by commodity $k \in K$.

$$\min \sum_{k\in K} \sum_{(i,j)\in A} c_{ij}^k y_{ij}^k \tag{2.4}$$

s. t.

$$\sum_{\{j:(i,j)\in A\}} y_{ij}^k - \sum_{\{j:(j,i)\in A\}} y_{ji}^k = b_i^k \qquad \forall k \in K, \forall i \in N, \tag{2.5}$$

$$l_{ij} \leq \sum_{k\in K} y_{ij}^k \leq u_{ij} \qquad \forall (i,j) \in A, \tag{2.6}$$

$$l_{ij}^k \leq y_{ij}^k \leq u_{ij}^k \quad \forall k \in K, \forall (i,j) \in A, \tag{2.7}$$

$$y_{ij}^k \in \mathbb{Z}^+ \quad \forall k \in K, \forall (i,j) \in A. \tag{2.8}$$

We might formulate a variety of alternative multicommodity models with different assumptions. In the model above, we also impose individual flow bounds for each commodity (constraints (2.7)).

Models for integral multicommodity flow problem must impose integrality on the flow variables to obtain integral solutions (see constraints (2.8)). Garey and Johnson [1979] prove that this problem is NP-hard if there are at least two commodities.

2.3.2 Set Partitioning Problem

The *set partitioning problem* is a 0-1 integer programming problem that determines a minimum cost partitioning of the elements of a set M into feasible smaller subsets. In this way, each element of M must be contained in one, and only one, those subsets.

Let M be a finite set and N be a finite family of feasible subsets of M. A constraint set defines N. Moreover, each subset $j \in N$ has an associate cost c_j. Let x_j be a binary decision variable, where x_j is equal 1 if the subset $j \in N$ is part of the solution and 0 otherwise. Let a_{ij} be 1 if subset $j \in N$ contains element $i \in M$ and 0 otherwise. Next, we present the model for the set partitioning problem.

$$\min \sum_{j\in N} c_j x_j \tag{2.9}$$

s. t.

$$\sum_{j\in N} a_{ij} x_j = 1 \quad \forall i \in M, \tag{2.10}$$

$$x_j \in \{0,1\}, \quad \forall j \in N. \tag{2.11}$$

As proved in Garey and Johnson [1979], this problem is NP-hard. There are two main types of solution algorithms for the set partitioning problem in the literature: dual-fractional and primal algorithms [Tahir et al., 2019].

Many scheduling problems, such as bus, airline, and railway crew schedulings, are traditionally formulated as a set partitioning problem (e.g., Balas and Padberg [1975]; Marsten and Shepardson [1981]; Desaulniers et al. [1997]; Mingozzi et al. [1999]; Haase et al. [2001]; Mesquita and Paias [2008]; Perumal et al. [2021]).

Chapter 3

Multiple-Depot Vehicle and Crew Scheduling Problem

This chapter describes our approach to tackle the MDVCSP. It is the central part of this thesis, and the subjects in Chapter 2 are its basis.

Section 3.1 introduces the MDVCSP and presents the constraints and assumptions that we consider to solve a real-world problem in Brazil and a problem addressed in the literature. Section 3.2 discusses the modeling approaches proposed by Steinzen et al. [2010] and by us for the MDVCSP. Section 3.3 describes the ILS-MDVCSP matheuristic algorithm we propose to solve the MDVCSP and the underlying procedures and strategies.

3.1 Problem Definition

In the *multiple-depot vehicle and crew scheduling problem* (MDVCSP), vehicle and crew schedules are defined simultaneously and must be mutually compatible. That is, the workday of some crew must cover each operational activity of a vehicle. Besides, we consider more than one depot for fleet and labor management. Therefore, we must also assign each vehicle and crew on the schedule to a single depot.

The rest of this section presents the concepts involved in defining the MDVCSP.

Timetable trips is constituted by the set of trips to be made daily by the public bus transit company. Table 3.1 represents part of a timetable with trips of *line* A-B (a route over which a bus travels in both directions: A→B and B→A). In this table, arranged in columns and appearing sequentially, are the features of the trips considered relevant for the operational planning, they are: trip number (an identifier), start time of the trip, start point (which corresponds to the place of the start of the trip), end time of the trip, and end point (which refers to the point the trip ends).

Deadhead is each travel of a vehicle, which is not a timetable trip. According to its operational itinerary, this journey can be necessary to place a vehicle in an appropriate

Table 3.1: Example of part of a timetable

Trip Number	Start Time	Start Point	End Time	End Point
1	06:26	A	07:22	B
2	09:06	A	09:55	B
3	13:06	A	13:59	B
4	17:06	A	17:59	B
5	19:46	A	20:31	B
6	06:50	B	07:46	A
7	08:10	B	09:06	A
8	11:30	B	12:19	A
9	17:30	B	18:23	A
10	20:10	B	20:55	A

local. For example, we have the vehicle's travel to the starting point of a trip or return to the depot at the end of the day. Thus we must provide the deadhead times between the various stopping points of the vehicles. We organized this information in a structure called *deadhead matrix*.

Table 3.2 depicts a deadhead matrix with one depot G1 and the points A, B, C, and D. In this table, travel times are in minutes. As in this example, in most cases, this matrix is symmetric. Then, the travel time from X to Y is equal to the travel time from Y to X for any points X and Y. Furthermore, the distance from X to X is always 0.

Table 3.2: Example of a deadhead matrix

	G1	A	B	C	D
G1	0	42	41	55	55
A	42	0	32	15	24
B	41	32	0	32	22
C	55	15	32	0	15
D	55	24	22	15	0

Therefore, given a timetable and a deadhead matrix, the *vehicle scheduling problem* (VSP) consists of determining the operational routine for a vehicle fleet. The aim here is to make the execution of all trips feasible and, at the same time, to minimize the costs involved.

Vehicles itineraries are specified when addressing the VSP. Each itinerary corresponds to the trips assigned to a vehicle that can be carried out successively, without violating the operational rules and time and space limitations. It is noteworthy that, as long as it is feasible, a vehicle can return to the depot more than once during its operating day. Thus, the trips comprised between an exit/return sequence to the depot constitute the so-called *vehicle block*. The cost of the vehicle schedule includes fixed costs for each vehicle's purchase and variable costs for deadhead and waiting time outside the depot.

In the context of public bus transport, a *depot* corresponds to an installation for the management, maintenance, and parking of vehicles when they are not in use (operation). A depot can have an associated maximum capacity and type of fleet. Furthermore, a depot is located at a known distance from each trip's start and end point. Regarding the number of available depots, the VSP can be classified into *single-depot vehicle scheduling problem* (SDVSP) and *multiple-depot vehicle scheduling problem* (MDVSP).

Whenever a vehicle is out of the depot, there must be a *crew* responsible for it. Usually, a crew is composed of a driver and a collector. Therefore, the *crew scheduling problem* (CSP) consists of creating the workdays, so-called *duties*, for the crews. These duties must ensure the feasibly of the vehicle schedule. Besides, we must carry out this distribution of work to minimize labor costs and, at the same time, comply with labor legislation, collective labor agreements, and the operational rules under which the company operates.

Therefore, to address the CSP, the vehicle itineraries from a solution for the VSP are considered. We often observe that a vehicle's operational routine is not adequate to be executed entirely by a single crew (for example, because of its long duration). However, changing the crew responsible for a vehicle cannot be done at any time. It will only occur at so-called *relief points*, that is, at appropriate locations and time intervals.

Whereas in the VSP, the manipulated unit is the trip, in the CSP, it is the *task*. A task represents the smallest amount of work that can be assigned to a crew. It includes consecutive trips of the same vehicle and between which there is no relief point.

In addition to the tasks, many studies propose the formation of the so-called *piece of work* [Haase et al., 2001; Freling et al., 2003; Huisman et al., 2005; Steinzen et al., 2010] to solve the problem. Each piece of work includes consecutive tasks of a vehicle, is limited by a minimum and a maximum duration, must be fully assigned to just one crew, and does not include time for rest/feeding of the crew. In this case, a duty consists of combining pieces of work.

To meet legal and operational requirements, companies usually define the types of duties that are valid. Some characteristics that can be considered to specify the types of duties are minimum/maximum working time, minimum break length, maximum time of overtime, and time allowed for the beginning and end of a duty.

Figure 3.1 shows the itinerary of a vehicle and its decomposition in a duty. The vehicle makes five trips and returns to the depot once during its itinerary. Thus, it consists of two vehicle blocks. The vehicle's itinerary is decomposed into four tasks as there are only two relief points, station C and DEPOT. From these tasks, we created three feasible pieces of work. We also define a crew's duty, considering that a feasible duty is formed by, at most, two pieces of work and that there must be a break between them. Note that while the vehicle is parked at the depot, the crew takes a break for rest/feeding.

From the above, the MDVCSP approaches the MDVSP and CSP problems simul-

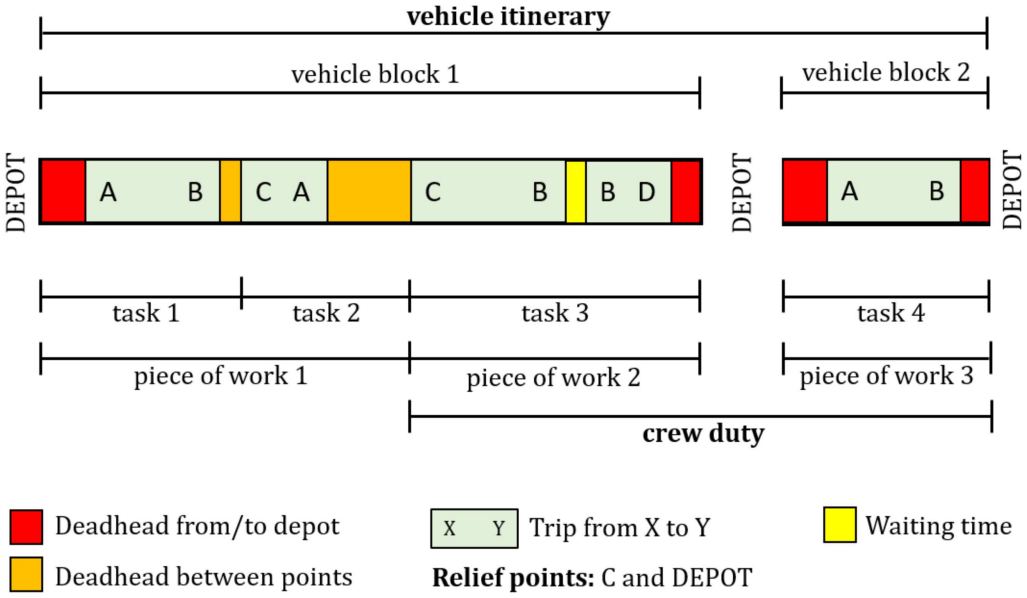


Figure 3.1: Example of vehicle itinerary and its crew duty.

taneously.

This thesis proposes a matheuristic capable of satisfactorily addressing real-world and large-scale instances for the MDVCSP. These instances come from the public bus transport system in the city of Belo Horizonte, MG. It is a Brazilian city with the fourth largest fleet of buses in the country.

The constraints considered in solving the problem are:

C1 - Each timetable trip can be serviced by any depot;

C2 - Each trip has exactly two relief points: one at the beginning and one at the end of the trip. That is, each trip of the vehicle is considered a task;

C3 - Each vehicle is assigned to a single depot and must start and end your itinerary there;

C4 - Each crew is associated with a depot and can only drive vehicles from it. Every duty does not need to start and end in the depot. However, when they take place outside the depot, these activities require additional time for the crew to prepare;

C5 - The viability of a piece of work depends only on its duration, which is limited by a minimum and a maximum time;

C6 - During its itinerary, the vehicle will return to the depot whenever the idle time between two consecutive trips is sufficient to perform a round trip to the depot;

C7 - A crew can change vehicles (*changeover*), but only during a rest/feeding break (that is, between pieces of work);

C8 - The so-called *continuous attendance* is respected. That is, always there is a crew responsible for a vehicle that is outside the depot, regardless of whether it is stationary or moving;

C9 - Each depot is unlimited concerning the number of associated crews;

C10 - Each depot has a limited number of assigned vehicles;

C11 - If the rest/feeding interval of the crew occurs between work shifts, the crew must end the first shift at the start point of the second shift;

C12 - Each vehicle must remain in the depot for a minimum of time at the end of its daily operating itinerary. This time is for cleaning and maintaining the vehicle.

We generate an initial solution for the MDVCSP addressing its subproblems, MDVSP and CSP, sequentially. Initially, we solve the MDVSP in optimality. Then, from its solution with n depots ($n \geq 2$), we generate n independent CSPs, that is, one CSP per depot. Each CSP is solved separately by a procedure that generates an initial solution and a B-VND local search heuristic. Note that it does not make sense to solve a single CSP considering all depots simultaneously because, according to constraints C4, each crew can only drive vehicles from the same depot.

From this initial solution, we approach the MDVSP and CSPs in an integrated way, i.e., the MDVCSP. Iteratively, we make changes (perturbations) in the solution for the MDVSP. If the itinerary of a vehicle is modified, its pieces of work must be rebuilt (see Figure 3.1). So, in the associated CSPs, we must remove the pieces of work that no longer exist and assign each new piece of work to some crew. In this way, we will be able to obtain lower total cost vehicle and crew schedules and, at the same time, keep them mutually compatible and feasible.

Table 3.3 summarizes the principal terms introduced in this section.

3.2 Modeling Approach

This section describes the modeling approaches for the MDVCSP. Subsections 3.2.1 and 3.2.2, respectively, present the mathematical formulations proposed by Steinzen et al. [2010] and by us for the MDVCSP. These formulations are composed of two parts:

1. A multicommodity network flow formulation based on a time-space network for the MDVSP;

2. A set partitioning formulation for the CSP.

Table 3.3: Concepts involved in defining the MDVCSP

Term	Meaning
Timetable trips	It is constituted by the set of trips to be made daily by the public bus transit company.
Deadhead	It is each travel of a vehicle, which is not a timetable trip.
Deadhead matrix	The structure that provides the deadhead times between the various stopping points of the vehicles.
Vehicle itinerary	Corresponds to the trips assigned to a vehicle that can be carried out successively without violating the operational rules and time and space limitations.
Vehicle block	The trips of a vehicle comprised between an exit/return sequence to the depot.
Depot	Installation for the management, maintenance, and parking of vehicles when they are not in use (operation).
Crew	It is usually composed of a driver and a collector responsible for each vehicle's activity.
Duty	Consists of a crew's workday.
Relief point	Appropriate location and time interval for changing the crew responsible for a vehicle.
Task	Represents the smallest amount of work that can be assigned to a crew. It includes consecutive trips of the same vehicle and between which there is no relief point.
Piece of work	Includes consecutive tasks of a vehicle, is limited by a minimum and a maximum duration, must be fully assigned to just one crew, and does not include time for rest/feeding of the crew. Many studies propose the formation of pieces of work as a solution strategy for the CSP.
VSP	Vehicle scheduling problem.
SDVSP	Single-depot vehicle scheduling problem.
MDVSP	Multiple-depot vehicle scheduling problem.
CSP	Crew scheduling problem.
MDVSP	Multiple-depot vehicle and crew scheduling problem.

3.2.1 A Literature Modeling Approach for the MDVCSP

This subsection presents the mathematical formulation proposed by Steinzen et al. [2010] for the MDVCSP.

To model the problem, Steinzen et al. [2010] consider the same constraints presented in Section 3.1, except constraints C11 and C12. Regarding constraint C11, they consider that there may be a change of point in a rest/feeding interval, but there must be enough time for the crew to comply with the rest time and walk between these locations. About constraint C12, there is no limitation on a vehicle's itinerary duration.

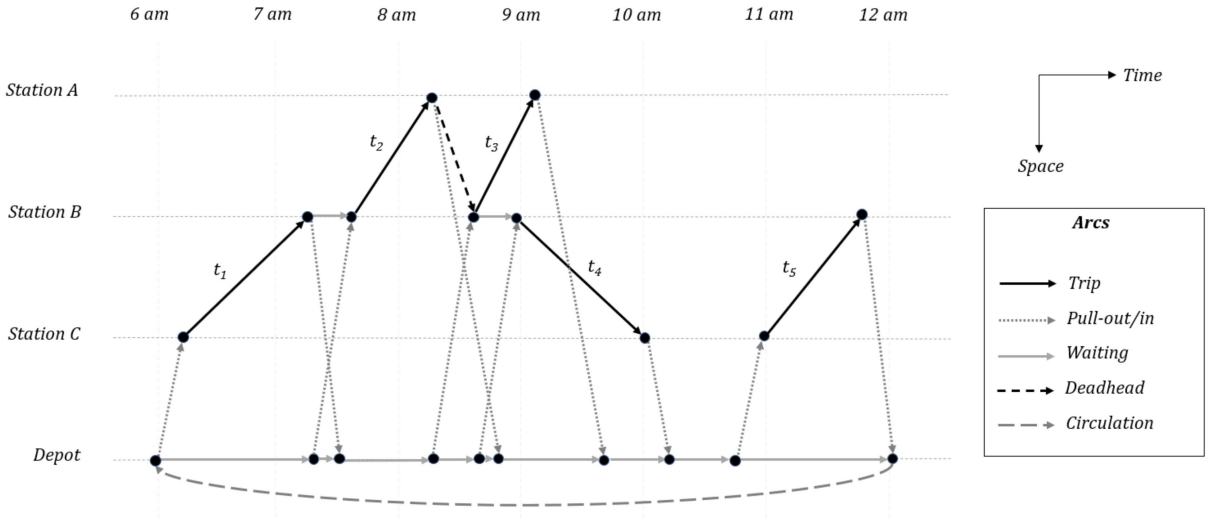


Figure 3.2: Time-space network layer for the MDVSP.

3.2.1.1 Time-Space Network for the MDVSP

In a *time-space network* for the MDVSP each node represents a *location* in a given *instant*; each arc corresponds to a transition in time and, possibly, in space. For each vehicle stopping point (station or depot), we define an imaginary timeline with nodes representing the departure and arrival events at this point. In the timeline, the nodes are increasingly ordered by the instant they represent.

We associate a network layer with each depot of the problem. Figure 3.2 illustrates a network layer with three stations (A, B, and C), five trips, and the respective depot. In this example, the planning horizon is from 6:00 am to 12:00 am. The types of arcs present in the network are:

1. *trip arc*: arc associated with a trip. It connects the departure node (station and start time of the trip) to the arrival node (station and end time of the trip);

2. *pull-out/pull-in arcs*: a *pull-out arc* connects a node in the depot to the starting node of a trip and represents the deadhead from the depot to the start station of the trip. A *pull-in arc* connects the arrival node of a trip to a node in the depot and represents the deadhead from the end station of the trip to the depot;

3. *waiting arc*: connects consecutive nodes on the point's timeline and corresponds to waiting at the point. We eliminate *waiting arcs* from the network that represents long durations in the stations. Note that in Figure 3.2, for example, there is no *waiting arc* in station C connecting the trips t_4 and t_5. This simplification is made possible by the constraint C6 of Section 3.1, initially proposed by Huisman et al. [2005] to address a problem in the literature;

4. *deadhead arc*: connects the arrival node of a trip to a departure node in another station where there are trips compatible with that trip. Therefore, this arc represents a deadhead between compatible trips. In Figure 3.2, to execute trips t_2 and t_3 consecutively, a vehicle needs to make a deadhead from station A to station B. On the other hand, to execute trip t_4 just after t_2, the vehicle needs to make the same previous deadhead and must remain to wait in station B;

5. *circulation arc*: connects the last node (*sink*) to the first node (*source*) in the depot timeline. This arc allows flow circulation in the network. Each flow unit in the *circulation arc* corresponds to a vehicle used in the schedule.

To the best of our knowledge, the formulations proposed in the literature for the MDVCSP represent the associated MDVSP in two ways: 1– *connection-based network*, initially proposed by Huisman et al. [2005]; or 2– *time-space network* [Steinzen et al., 2010], as presented above.

In a connection-based network, we have an arc between each pair of compatible trips. On the other hand, many of these connections are considered only implicitly in the time-space network. In Figure 3.2, for example, the trips t_2 and t_4 are compatible since it is possible to traverse different types of arcs from t_2 and arrive at t_4.

As shown in Kliewer et al. [2006] and Steinzen et al. [2010], in the time-space network the amount of *deadhead arcs* is much less than in the connection-based network. This fact has a substantial impact on the number of variables and constraints of the model for the MDVSP. So, we choose to use the time-space network to formulate the vehicle scheduling in this thesis.

For a detailed description about building time-space networks, see Kliewer et al. [2006] and Steinzen [2007b].

3.2.1.2 Mathematical Formulations for the MDVSP and MDVCSP

Let $T = \{1, 2, ..., m\}$ be the set of all m timetable trips and $D = \{1, 2, ..., n\}$ be the set of n depots. For each depot $d \in D$ there is an acyclic time-space network layer $G^d = (N^d, A^d)$, where N^d is the set of nodes and A^d is the set of arcs. We denote $\tilde{A}^d \subset A^d$ as the set of arcs that represent activities that involve the joint participation of vehicle and crew. So, $\tilde{A}^d$ does not include the *waiting arcs* in the depots, according to the *continuous attendance* requirement (Section 3.1, constraint C8). Let $A^d(t) : T \rightarrow A^d$ be the function that returns a set with precisely one *trip arc*, $(i, j) \in A^d$, associated with the trip $t \in T$, if t can be assigned to the depot $d \in D$. Otherwise, we have an empty set. The maximum

capacity u_{ij}^d of each arc $(i,j) \in A^d, \forall d \in D$, is defined as follows: *pull-out/in* and *trip arcs* have a maximum capacity of 1; all other arcs have a maximum capacity equal to the number of vehicles available in the depot $d \in D$. Each arc $(i,j) \in A^d$ has an associated cost c_{ij}^d. In *circulation arcs*, c_{ij}^d represents the cost of purchasing a vehicle. In the other arcs, c_{ij}^d is the operating cost. Let K^d be the set of duties associated with the depot $d \in D$. $K^d(i,j) \subset K^d$ is the set of duties associated with the depot d that covers the $(i,j) \in \tilde{A}^d$ arc. f_k^d refers to the cost of the duty $k \in K^d$ and involves fixed and working time crew costs.

Be the following types of decision variables:

- y_{ij}^d: indicates the flow associated with the arc $(i,j) \in A^d$, $d \in D$ (flow variable);
- x_k^d: determines whether the duty $k \in K^d$, $d \in D$, is part of the schedule (binary variable).

The model proposed by Steinzen et al. [2010] for the MDVCSP is presented below.

$$\min \sum_{d \in D} \sum_{(i,j) \in A^d} c_{ij}^d y_{ij}^d + \sum_{d \in D} \sum_{k \in K^d} f_k^d x_k^d, \tag{3.1}$$

s. t.

$$\sum_{d \in D} \sum_{(i,j) \in A^d(t)} y_{ij}^d = 1 \quad \forall t \in T \tag{3.2}$$

$$\sum_{\{j:(j,i) \in A^d\}} y_{ji}^d - \sum_{\{j:(i,j) \in A^d\}} y_{ij}^d = 0 \quad \forall d \in D, \forall i \in N^d \tag{3.3}$$

$$\sum_{k \in K^d(i,j)} x_k^d - y_{ij}^d = 0 \quad \forall d \in D, \forall (i,j) \in \tilde{A}^d \tag{3.4}$$

$$0 \leq y_{ij}^d \leq u_{ij}^d, y_{ij}^d \in \mathbb{N} \quad \forall d \in D, \forall (i,j) \in A^d \tag{3.5}$$

$$x_k^d \in \{0,1\} \quad \forall d \in D, \forall k \in K^d. \tag{3.6}$$

The objective is to minimize the cost of vehicle and crew schedules (3.1). Constraints (3.2) ensure that each timetable trip will be assigned to precisely one vehicle from one of the depots that can serve it. In constraints (3.3), flow conservation at the nodes is ensured for each network layer (depot). Constraints (3.4) guarantee that, for each depot, there will be the same number of vehicles and crews covering each activity (arc) that requires the association of a vehicle and a crew. Constraints (3.5) ensure that the maximum capacities of the flow variables are respected.

Given the constraints C1 to C12 of Section 3.1, we consider C6 and C12 (when used) in defining the time-space network, and C2, C4, C5, C7, and C11 (when used) in constructing the feasible duties sets ($K^d, \forall d \in D$). Finally, we associate the other constraints of the problem directly to the model equations, as follows: C1 - Equation (3.2),

C3 - Equation (3.3), C8 - Equation (3.4), C10 - Equation (3.5), and C9 - Equation (3.6) (that is, we only specify the domain of variables x_k^d, and we do not limit the number of variables with value 1).

In the formulation for MDVCSP (3.1)–(3.6), considering only the portion of the objective function associated with vehicle schedule ($\sum_{d \in D} \sum_{(i,j) \in A^d} c_{ij}^d y_{ij}^d$) and the constraints (3.2), (3.3), and (3.5), we get a formulation for the MDVSP. This formulation was proposed in Kliewer et al. [2006].

In this formulation, an optimal solution (flow) represents a set of optimum vehicle schedules due to the aggregation of connections in the time-space network specification. Figure 3.3 illustrates this situation in a small portion of the network and its optimal flow. There are two different ways to sequence trips t_1, t_2, t_3, and t_4 on the itineraries of two vehicles, they are: $t_1 - t_3$ (vehicle 1) and $t_2 - t_4$ (vehicle 2) or $t_1 - t_4$ (vehicle 1) and $t_2 - t_3$ (vehicle 2). Thus, we use a flow decomposition procedure to obtain a specific vehicle schedule. There are different strategies for defining paths in the time-space network with the optimal flow. Each path from the source node to the sink node in the depot timeline represents a vehicle itinerary.

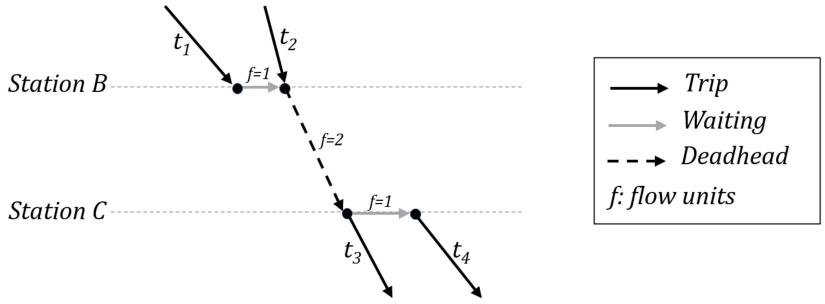


Figure 3.3: Flow decomposition possibilities.

3.2.2 Proposed Modeling Approach for the MDVSP and MDVCSP

In this subsection, we discuss our network structure and mathematical formulation for the MDVSP that considers the particularities of the Brazilian real-world problem addressed in the present work. That is, this modeling approach meets the assumptions and constraints stated in Section 3.1. As we will describe later, our MDVCSP resolution heuristic uses this model to generate an initial vehicle schedule.

3.2.2.1 Proposed Time-Space Network for the MDVSP

The formulation for the MDVSP presented in Subsection 3.2.1.2 is consistent and efficient for solving instances widely used in the literature, as the ones proposed by Huisman et al. [2005] and Steinzen et al. [2010]. However, it is not adequate to represent the Brazilian real-world problem that we have addressed in this thesis.

In the instances from literature, trips always start from 6 am and end before 1 am. Thus, in the 24-hours planning horizon, there is an interval of more than 5 hours in which no trip is performed. However, in the Brazilian real-world instances that we consider, there are timetable trips distributed throughout the day. There is even an extrapolation of the 24-hour planning horizon. In this sense, some trips start at the end of a day and end in the early hours of the next day; other trips start in the day's first moments. Besides, the public transport management company requires that every vehicle remains in the depot for a minimum time at the end of its daily itinerary. This time is used for cleaning and overhauling the vehicle. This situation probably occurs in many large companies in the world.

Figure 3.4 illustrates the timeline of a depot for a Brazilian real-world instance. Shaded nodes are associated with *pull-out* arcs (depot exit) and non-filled nodes with *pull-in* arcs (depot arrival). This example requires that the itinerary of any vehicle lasts a maximum of 23 hours and 30 minutes.

Suppose a single circulation arc in Figure 3.4 connects the last node in the depot's timeline to the first one. In that case, we could define itineraries that are longer than acceptable. So we created more than one circulation arc. Each arc will represent a valid operating period (shift) for the vehicles in this depot.

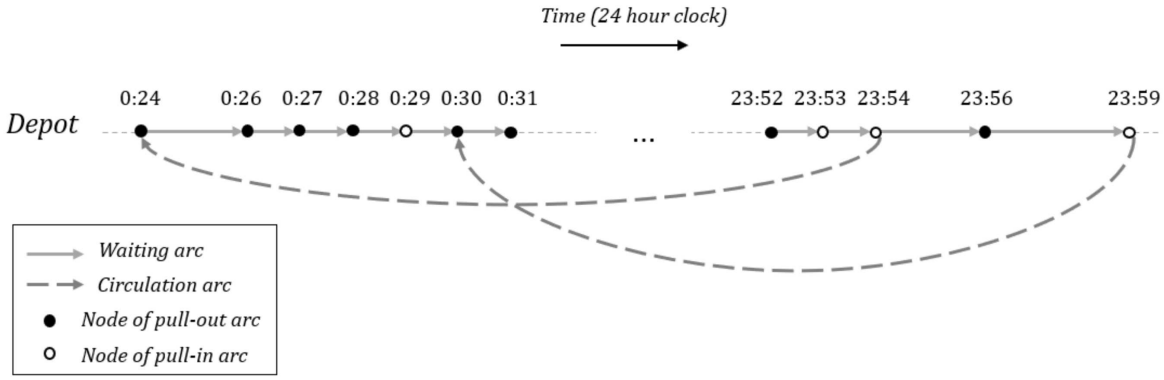


Figure 3.4: Circulation arcs of the timeline of a depot in a Brazilian real-world problem.

By definition and without loss of generality, we can consider that a circulation arc must always connect a node of an arc *pull-in* (arrival at the depot) to a node of an arc *pull-out* (exit of the depot). Thus, below we detail how the circulation arcs are created.

We go through the depot timeline iteratively (node by node) and from right to left, as follows: at each node of a *pull-in* arc reached (origin of the circulation arc), we search for the node of a *pull-out* arc (destination of the circulation arc). We chose this second node in such a way as to define the longest feasible operating period possible. When selecting the destination node of the circulation arc, two situations can occur:

1. There is already a circulation arc arriving at this node. So, we did not create a new circulation arc;

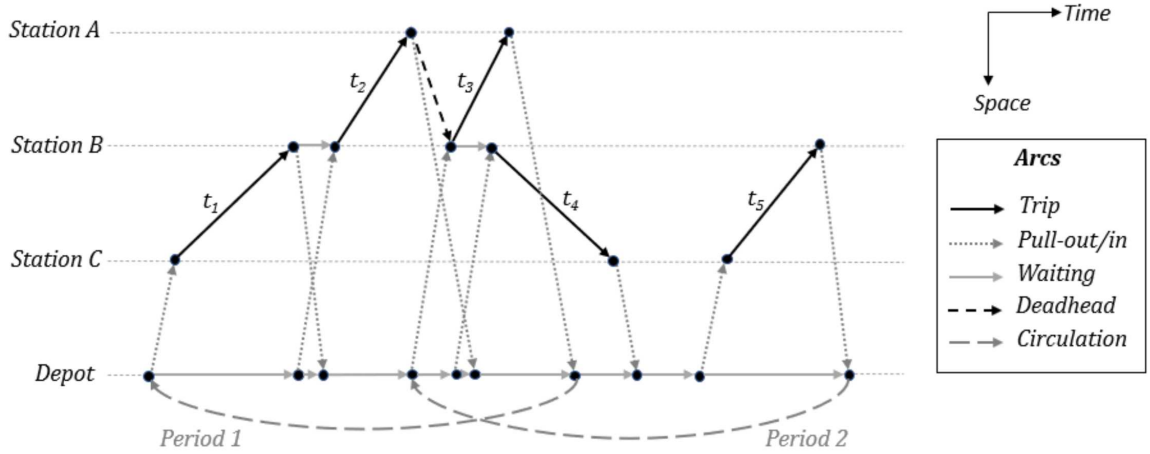
2. There is no circulation arc with a destination at this node. So, we created a new circulation arc. If the target node of this arc is the first node in the depot timeline, we finish the arc definition process.

At the end of this procedure, all nodes in the depot will be covered, i.e., they will belong to one or more operating periods. Furthermore, this procedure creates all the viable operating periods.

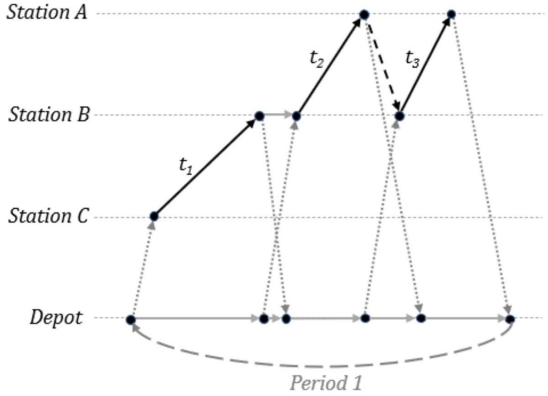
In our representation of the MDVSP, we must maintain a circulation arc per network layer. Therefore, each layer will be associated with the depot-operating period feasible combination; and it will only represent trips that can be carried out in the depot and period considered. In Figure 3.5, we break down the network layer of a depot with two circulation arcs in Figure 3.5a (inconsistent representation) into two network layers in Figures 3.5b and 3.5c. Therefore, in general, the timeline of a depot with w circulation arcs will give rise to w layers of the time-space network.

To construct the time-space network, we follow Kliewer et al. [2006]. We represent each network layer by an adjacency list and define the network arcs in the order: 1− trip arcs, 2− pull-out/pull-in arcs, 3− deadhead arcs, 4− waiting arc, and 5− circulation arc. According to Kliewer et al. [2006], we can connect each pair of compatible trips through a deadhead arc. However, the number of deadhead arcs can be high and implies a problem size which cannot be handled by state-of-the-art solvers. Thus, we used the modeling technique described in Kliewer et al. [2006] to aggregate deadhead arcs between trips. This aggregation is composed of three stages and provides a crucial model reduction. Furthermore, we used the proposed procedure above to define each circulation arc and build each network layer for the Brazilian real-world instances.

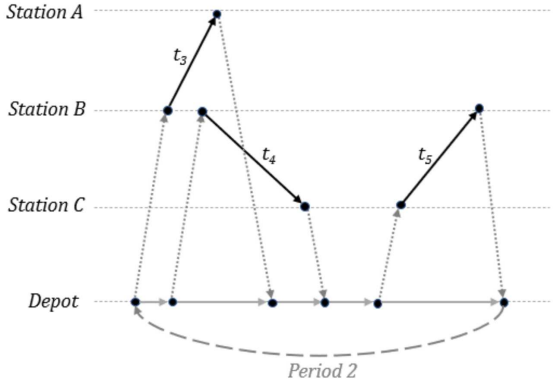
In the following, we discuss the complexity of the proposed network structure. Let m be the number of timetable trips, p the number of different stations, and ℓ the number of network layers (depot-operating period combinations). Thus, the number of *deadhead arcs* in a network layer is $O(mp)$ because one *deadhead arc* connects a trip with all subsequent trips at a different station. On the other hand, the number of *waiting arcs*, *pull-out/pull-in arcs*, or *trip arcs* grows linearly with the number of trips in a network layer. Finally, there is one circulation arc in a network layer. Therefore, the number of



(a) Network layer of a depot with two circulation arcs.



(b) Network layer with only the circulation arc of period 1.



(c) Network layer with only the circulation arc of period 2.

Figure 3.5: Definition of one network layer for each depot-period.

arcs in the time-space network is $O(\ell mp)$. Note that this number of arcs determines the number of variables of the proposed mathematical formulation for the MDVSP, as shown in the next subsection.

3.2.2.2 Proposed Mathematical Formulations for the MDVSP and MDVCSP

The formulation we propose for the MDVSP is presented below (3.7)–(3.11).

Let $T = \{1, 2, ..., m\}$ be the set of all m timetable trips and $D = \{1, 2, ..., n\}$ be the set of n depots. From the depots in D we define $\Delta = \{1, 2, ..., \ell\}$, the set of all possible depot-operating period combinations. Each feasible operating period is defined by a circulation arc. For each depot-period $\delta \in \Delta$ we have an acyclic time-space network layer

$G^\delta = (N^\delta, A^\delta)$, where: N^δ is the set of nodes and A^δ the set of arcs. Let $A^\delta(t) : T \to A^\delta$ be the function that returns a set with exactly one *trip arc*, $(i, j) \in A^\delta$, associated with the trip $t \in T$, if t can be covered by the depot and operating period given by $\delta \in \Delta$. Otherwise, we have an empty set.

Let Δ^d be the set of depot-period combinations in which the depot involved is $d \in D$. F^δ is the set formed by the circulation arc associated with the depot-period $\delta \in \Delta$. cap^d refers to the capacity of the depot $d \in D$ concerning the number of vehicles.

Each arc $(i, j) \in A^\delta$ has an associated cost c^δ_{ij}. In *circulation arcs*, c^δ_{ij} represents the cost of purchasing a vehicle. In the other arcs, c^δ_{ij} is the operating cost. Finally, we have the decision variable y^δ_{ij} which indicates the flow associated with the arc $(i, j) \in A^\delta$, $\delta \in \Delta$.

$$\min \sum_{\delta \in \Delta} \sum_{(i,j) \in A^\delta} c^\delta_{ij} y^\delta_{ij}, \tag{3.7}$$

s. t.

$$\sum_{\delta \in \Delta} \sum_{(i,j) \in A^\delta(t)} y^\delta_{ij} = 1 \quad \forall t \in T \tag{3.8}$$

$$\sum_{\{j:(j,i) \in A^\delta\}} y^\delta_{ji} - \sum_{\{j:(i,j) \in A^\delta\}} y^\delta_{ij} = 0 \quad \forall \delta \in \Delta, \forall i \in N^\delta \tag{3.9}$$

$$\sum_{\delta \in \Delta^d} \sum_{(i,j) \in F^\delta} y^\delta_{ij} \leq cap^d \quad \forall d \in D \tag{3.10}$$

$$y^\delta_{ij} \in \mathbb{Z}^+ \quad \forall \delta \in \Delta, \forall (i, j) \in A^\delta. \tag{3.11}$$

The objective is to minimize the cost of vehicle schedule (3.7). The constraints (3.8) ensure that each timetable trip will be assigned to precisely one vehicle. This vehicle must be linked to a depot-operating period combination that is compatible with the trip. In (3.9) we have the flow conservation constraints at the nodes of each network layer (depot-period). The constraints (3.10) ensure that the capacity of each depot, in relation to the number of vehicles, is respected. The constraints (3.8)–(3.11) implicitly establish an upper limit for the value of each flow variable y^δ_{ij}. Let u^δ_{ij} be the upper limit for $(i, j) \in A^\delta$ and $\delta \in \Delta$, we have: $u^\delta_{ij} = 1$, for *trip* and *pull-in/out arcs*. In addition, u^δ_{ij} depends on the depot capacity associated with δ. That is, the sum of the flows of the *circulation arcs* of a depot cannot exceed the capacity of the depot; and the flow in any arc in a network layer cannot be greater than the flow of your *circulation arc*.

This formulation can be combined with the set partitioning formulation for the CSP of Subsection 3.2.1.2. Thus, we obtain a formulation for the MDVCSP.

3.3 Solution Approach

Given the difficulty of solving real-world instances of the MDVCSP using exact mathematical methods, we propose a matheuristic algorithm for solving it. This matheuristic algorithm combines two strategies into an ILS-based framework: a branch-and-bound algorithm for solving the MDVSP and a VND-based algorithm for treating the associated CSPs.

Next, we describe how to represent the MDVSP, CSP, and MDVCSP in Subsection 3.3.1. In Subsection 3.3.2, we present the neighborhood structures used to explore the solution space of these problems. In Subsection 3.3.3, we define the evaluation functions to vehicle and crew schedules. Subsection 3.3.4 presents the proposed matheuristic algorithm for solving the MDVCSP. Subsection 3.3.5 describes heuristic procedures to treat the CSP exclusively, which we use as auxiliary methods to solve the MDVCSP in Subsection 3.3.4.

3.3.1 Solution Representation

3.3.1.1 Solution Representation for the MDVSP

A solution s^v for the MDVSP consists of a list of vehicles. In turn, for each vehicle, we associate its depot and the list of trips that it will perform during a working day. Figure 3.6 illustrates a solution s^v for the MDVSP in which a fleet of three vehicles, distributed over two depots, must perform ten trips.

In this representation, we organize the trips made by each vehicle according to their start times. Thus, it is possible to evaluate the vehicle's operational itinerary in several aspects, such as the locomotion time out of operation (deadhead), the waiting time at the station between two consecutive trips, the number of returns to the depot, and the viability of the vehicle block.

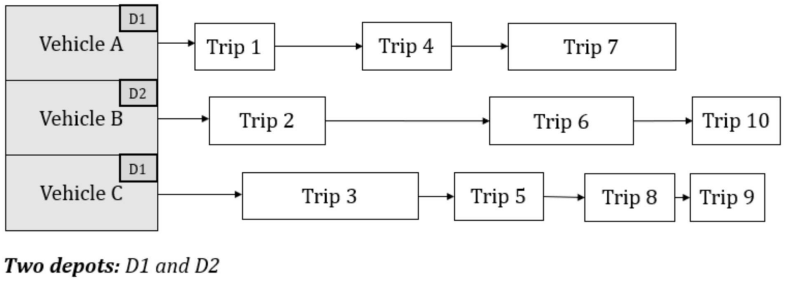


Figure 3.6: Example of a solution s^v for the MDVSP.

3.3.1.2 Solution Representation for the CSP

A solution s^c for the CSP consists of a list of duties. Each duty associates the pieces of work to be performed by the same crew during a working day. Figure 3.7 illustrates a solution s^c for the CSP. In this example, there is a depot with three duties and four pieces of work assigned.

We keep the pieces of work of each duty on a list and sort them in ascending order by their start times. In this way, it is possible to determine relevant characteristics of the duty, such as time reserved for rest/food, the occurrence of vehicle change, and the existence of an overlap between tasks.

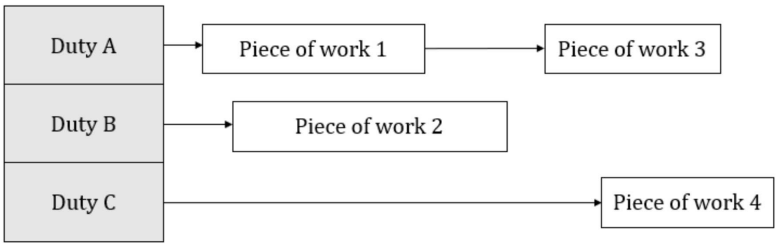


Figure 3.7: Example of a solution s^c for the CSP.

3.3.1.3 Solution Representation for the MDVCSP

We represent a solution S^{vc} for the MDVCSP by a pair $S^{vc} = [s^v, S^c]$, where s^v represents the solution to the MDVSP and S^c represents the solution set of the n CSPs associated to each solution s^v with n depots. Both representations were previously described and illustrated in Subsections 3.3.1.1 and 3.3.1.2, respectively.

Let s^v be a solution for the MDVSP, s_i^c a solution for the CSP associated with the i-th depot, and n the number of depots. Then, we define S^c as a set of n solutions, one

for each of the n CSPs obtained from s^v, that is:

$$S^c = \{s_1^c, s_2^c, ..., s_n^c\}\,. \tag{3.12}$$

As some depots may be inactive (i.e., without linked vehicles) in a solution for the MDVSP, some solutions s_i^c into the set S^c may be empty (i.e., without duties).

3.3.2 Neighborhood Structures

3.3.2.1 MDVSP Neighborhood Structures

We apply the six types of moves described below to explore the MDVSP solution space. Each move defines a neighborhood structure. Figure 3.8 illustrates all of them.

1. *Trip relocate without depot adjustment* (Neighborhood structure N_r^v): It consists of reallocating a trip without allowing changing depots. Figure 3.8b illustrates this move. A trip is transferred from one vehicle to another one, and the depots of the vehicles involved are maintained;

2. *Trip relocate with depot adjustment* (Neighborhood structure N_{rd}^v): It consists of reallocating a trip allowing changing depots. In Figure 3.8c, a trip is transferred from one vehicle to another one. This transfer seeks to assign each modified vehicle to the depot that implies the lowest operating cost considering the new itinerary configuration;

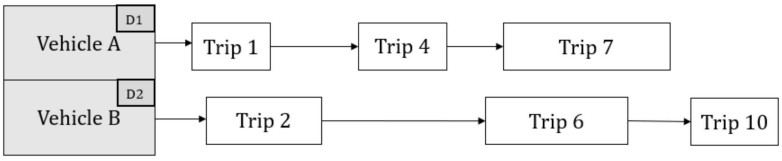
3. *Trip exchange without depot adjustment* (Neighborhood structure N_e^v): This move consists of exchanging trips between two vehicles, not allowing changing their depots. Figure 3.8d shows how it works. A vehicle exchanges one of its trips with a trip of another vehicle, and the depots of the vehicles involved are maintained;

4. *Trip exchange with depot adjustment* (Neighborhood structure N_{ed}^v): This move consists of exchanging trips between two vehicles allowing them to change their depots. Figure 3.8e illustrates its operation. A vehicle exchanges one of its trips with a trip from another vehicle. In this exchange, we seek to assign each modified vehicle to the depot that implies the lowest operating cost considering the new itinerary configuration;

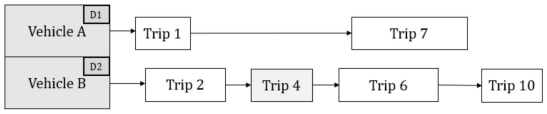
5. *Trip redistribution* (Neighborhood structure N_{dd}^{v}): This move consists of redistributing the trips of two vehicles allowing them to change their depots. Figure 3.8f) shows the application of this move. Two vehicles have all their trips removed. These trips are then randomly redistributed to each other. Finally, we seek to assign each vehicle to the depot that implies the lowest operating cost considering the new itinerary configuration;

6. *Depot change* (Neighborhood structure N_{dc}^{v}): It consists of changing the depot linked to a vehicle. Figure 3.8g shows a neighbor in which vehicle B changes from depot D2 to D1.

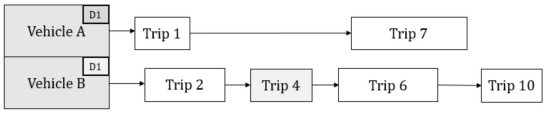
In the relocation, exchange, and redistribution moves, the vehicles involved should not necessarily belong to the same depot. Besides, we apply only those moves that maintain the feasibility of the solution.



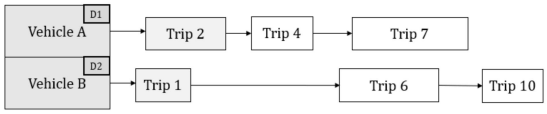
(a) Initial MDVSP solution.



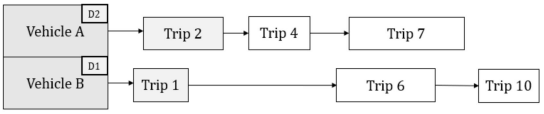
(b) Move 1 - Relocates the Trip 4 of Vehicle A to Vehicle B.



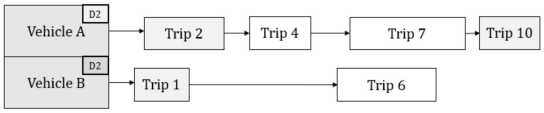
(c) Move 2 - Equal Figure 3.8b and makes depot adjustments.



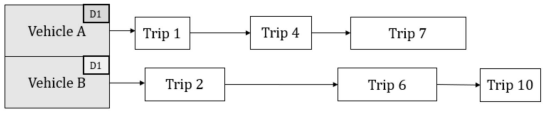
(d) Move 3 - Exchange of Trip 1 of Vehicle A with Trip 2 of Vehicle B.



(e) Move 4 - Equal Figure 3.8d and makes depot adjustments.



(f) Move 5 - Redistributes the trips of the A and B vehicles and makes depot adjustments.



(g) Move 6 - Vehicle B depot change from D2 to D1.

Figure 3.8: MDVSP moves.

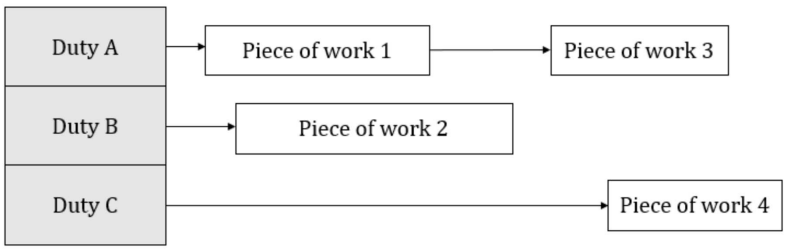
3.3.2.2 CSP Neighborhood Structures

We use two types of moves to explore the CSP solution space: relocate a piece of work and exchange pieces of work. These moves are illustrated in Figure 3.9 and are described below, together with the associated neighborhood structures.

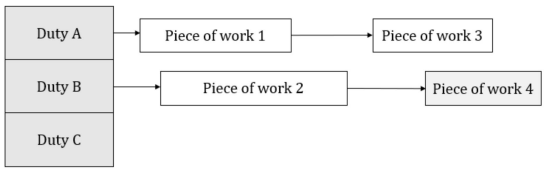
1. *Relocate piece of work* (Neighborhood structure N_r^c): It consists of reallocating one piece of work from one duty to another one. Figure 3.9b illustrates this move. A piece of work is transferred from one duty to another one;

2. *Exchange pieces of work* (Neighborhood structure N_e^c): This move consists of exchanging pieces of work between two duties. Figure 3.9c shows how it works. One piece of work belonging to a duty is exchanged with a piece of work from another duty.

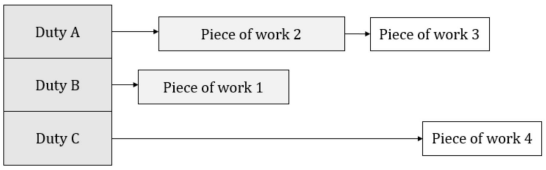
The manipulated duties belong to the same depot in these moves since there is an independent CSP per depot. Moreover, we only carry out movements that maintain the viability of the s^c solution.



(a) Initial CSP solution.



(b) Move 1 - Relocates the Piece of work 4 from Duty C to Duty B.



(c) Move 2 - Exchange of Piece of work 1 of Duty A with Piece of work 2 of Duty B.

Figure 3.9: CSP moves.

3.3.3 Evaluating Function

To determine the quality of a solution $S^{vc} = [s^v, S^c]$ for the MDVCSP, we associate a cost with vehicle and crew schedules.

We evaluate a solution s^v for the MDVSP based on the following function f^v (3.13), which should be minimized:

$$f^v(s^v) = totalVehicles \times vehicleCost + operationTime \times operationalCost, \quad (3.13)$$

where:

(a) *totalVehicles* is the number of vehicles used on the vehicle schedule;

(b) *vehicleCost* is the vehicle cost;

(c) *operationTime* is the total time, in minutes, that the vehicles in the fleet were out of the depot (regardless of the activities carried out during this period, which may include: trip, deadhead, or waiting at the station);

(d) *operationalCost* is the cost per minute due to a vehicle staying outside its depot.

In turn, we calculate the cost of a solution s^c for the CSP using the function f^c, as shown below in (3.14):

$$f^c(s^c) = totalCrews \times crewCost + totalWorkingTime \times workingTimeCost, \quad (3.14)$$

where:

(a) *totalCrews* is the number of crews on the schedule;

(b) *crewCost* is the cost of each crew;

(c) *totalWorkingTime* is the total working time, in minutes, of all crews on the schedule. That is, we consider the total duration of each duty, including the mandatory rest time;

(d) *workingTimeCost* is the cost of each minute of work for one crew.

Let $S^{vc} = [s^v, S^c]$ and $S^c = \{s^c_1, s^c_2, ..., s^c_n\}$ for n depots, the cost of a solution S^{vc} for MDVCSP is evaluated according to Equation (3.15):

$$f^{vc}(S^{vc}) = f^v(s^v) + \sum_{i=1}^{n} f^c(s^c_i), \quad (3.15)$$

where $f^v(s^v)$ and $f^c(s_i^c)$ are the functions that evaluate the solutions s^v and s_i^c for the MDVSP and CSP, respectively. There is no cost for infeasibility since the solution S^{vc} is kept feasible throughout the solution methods.

3.3.4 Matheuristic Algorithm for the MDVCSP

The matheuristic algorithm ILS-MDVCSP proposed for solving the MDVCSP combines an exact method with a heuristic method into an ILS-based framework. It solves the MDVSP optimally using a branch-and-bound algorithm. A constructive procedure and a B-VND local search procedure generate the solutions of the associated CSPs. Finally, the ILS-MDVCSP iteratively modifies the integrated solution through perturbations and performs a B-VND-based local search on the CSPs that have been changed.

The ILS-MDVCSP is described in Algorithm 4. Initially, we generate a solution s^v for the MDVSP (line 8). For that, an integer linear programming (ILP) optimization solver solves the problem from one of the formulations presented in the Subsections 3.2.1.2 and 3.2.2.2. The choice of the formulation depends on the MDVSP specifically addressed, that is, the MDVSP as defined in the literature or the Brazilian real-world problem addressed in this work.

As each depot considered in the MDVSP generates a CSP, in line 9 we define the set of solutions S^c. For a problem with n depots, S^c contains n crew schedules. This procedure is detailed in Algorithm 5.

In Algorithm 5, the itinerary of each vehicle of the solution s^v is partitioned into pieces of work to form the set P (line 4). Subsection 3.3.5.1 describes the two partitioning methods used: direct and inverse. For each vehicle in the schedule, we randomly choose one of these procedures. Then, we separate the pieces of work obtained by the depot (line 5). Thus, we generate n sets of pieces of work. When no vehicles are assigned to some available depots, some of them may be empty. In this way, there will be n CSPs to solve at most. At each iteration (lines from 8 to 12), a solution s_i^c for the CSP associated with the i-th depot is obtained and added to the solution set S^c. For that, we consecutively invoke the constructive procedure described in Algorithm 10 (Subsection 3.3.5.3) and the local search procedure based on the B-VND described in Algorithm 11 (Subsection 3.3.5.4).

We form a complete solution for the MDVCSP (S_*^{vc}) in line 10 of Algorithm 4. Then, we execute the procedures described in lines 12 to 14 as long as the maximum processing time ($time_{max}$) has not been reached.

Algorithm 6 describes the *perturbation* method. In this method, a perturbation of a certain level begins with applying randomly moves in the solution s^v for the MDVSP

(line 4).

We applied six levels of perturbations based on the moves defined in Subsection 3.3.2.1, namely:

1. *level 1*: one depot change move,
2. *level 2*: one trip relocation move,
3. *level 3*: one trip exchange move,
4. *level 4*: one trip redistribution move,
5. *level 5*: two trips exchange moves,
6. *level 6*: two trips redistribution moves.

As the perturbation level increases, the search procedure gradually moves away from the current solution towards other regions that have not yet been explored in the problem's solution space.

When carrying out the relocate and exchange moves of trips, we randomly chose between two possibilities: 1) maintaining the depots of the vehicles involved (neighborhood structures N_r^v and N_e^v) and 2) inserting the modified vehicles in the depots that generate lower cost (neighborhood structures N_{rd}^v and N_{ed}^v).

Any change in the MDVSP solution (s^v) causes changes to the CSP solutions (S^c). So, for vehicle and crew schedules to be kept individually feasible and mutually compatible, the pieces of work of the vehicles that have been modified must be rebuilt. For this, we randomly choose one of the partitioning methods defined in Subsection 3.3.5.1 (direct or inverse) for each vehicle. In this process, we must remove the pieces of work that no longer exist from the crews to which we assign them. We also need to assign each new piece of work to a crew of the schedule. Thus, the pieces of work removed and created are separated by depot (lines 5 and 6, respectively). Pieces of work are excluded from the CSP solution directly. To do it, we have only to find the crews responsible for these pieces (line 10). In lines 13 to 15, each new piece of work is inserted into the schedule using the heuristic procedure described in Algorithm 9 (Subsection 3.3.5.2). Due to the construction process, in set C_i, each vehicle's pieces of work are arranged sequentially. However, we can also sort this set by the start time of the pieces of work. Then, the parameter $sortPieces\_pert$ defines the order of the pieces of work in the set C_i (line 11).

From this perturbed solution for the MDVCSP, we apply the *LocalSearch* method (line 13 from ILS-MDVCSP algorithm). According to Algorithm 7, we apply a B-VND-based local search (Algorithm 11, Subsection 3.3.5.4) for solving each CSP that has been modified. In this step, the vehicle schedule is not changed.

Finally, we evaluate the new solution obtained at line 14 of the ILS-MDVCSP algorithm. According to Algorithm 8, we accept the new solution S^{vc} (vehicle and crew

schedules) only if it has a cost less than or equal to the cost of the current solution S_*^{vc}. Suppose there was an improvement in the solution. In that case, we restart the perturbation level to intensify the search in the current region. Otherwise, we increase the perturbation level by one unit to move away from the current region or restart it if it is already at the highest perturbation level.

Algorithm 4: ILS-MDVCSP

Data: T, the set of the timetable trips.
Data: n, the number of depots of the problem.
Data: l_{max}, the maximum level of perturbation.
Data: $time_{max}$, the maximum processing time.
Result: S_*^{vc}, the best solution found for the MDVCSP.
$time \leftarrow 0$ `// the current processing time of the algorithm`
$l \leftarrow 1$ `// the current level of perturbation`
$s^v \leftarrow solver(T, n)$
$S^c \leftarrow initialSolutions\_CSPs(s^v, n)$ `// See Algorithm 5`
$S_*^{vc} \leftarrow [s^v, S^c]$ `// The complete solution` S_*^{vc} `for the MDVCSP consists of the solution` s^v `for the MDVSP and the set` S^c `of solutions for the` n `CSPs`
while $time < time_{max}$ **do**
    $S_1^{vc} \leftarrow perturbation(S_*^{vc}, l)$ `// See Algorithm 6`
    $S_2^{vc} \leftarrow localSearch(S_1^{vc})$ `// See Algorithm 7`
    $[S_*^{vc}, l] \leftarrow acceptanceCriterion(S_*^{vc}, S_2^{vc}, l, l_{max})$ `// See Algorithm 8`

3.3.5 Heuristic Methods for the CSP

This subsection presents the heuristic procedures developed to address the CSP. These procedures consider that the vehicle schedule of an MDVSP depot is known. We apply them to solve the MDVCSP described in Subsection 3.3.4.

In Subsection 3.3.5.1, we show how to split the vehicle schedule into pieces of work. Then, in Subsection 3.3.5.2, we present the heuristic procedure for adding a new piece of work to a partial solution for the CSP. Finally, in Subsections 3.3.5.3 and 3.3.5.4, we detail the heuristic procedures for, respectively, generating an initial solution and performing a local search for the CSP.

Algorithm 5: *initialSolutions_CSPs*

Data: s^v, a solution for the MDVSP.
Data: n, the number of depots of the problem.
Result: S^c, the set of solutions for the n CSPs.
Let P be the set of pieces of work obtained from s^v
Let $\{P_1, P_2, ..., P_n\}$, where P_i is the set of pieces of work associated with the i-th depot and $P_i \subseteq P$
$S^c \leftarrow \{\}$
for $i \leftarrow 1$ **to** n **do**
    Let s_i^c be an initially empty solution for the CSP associated with the i-th depot
    if $P_i \neq \emptyset$ **then**
        $s_i^c \leftarrow P_i$ `// as described in Subsection 3.3.5.3, Algorithm 10`
        $s_i^c \leftarrow VND\_CSP(s_i^c)$ `// as described in Subsection 3.3.5.4, Algorithm 11`
    $S^c \leftarrow S^c \cup \{s_i^c\}$

3.3.5.1 Heuristic Methods for Generating the Pieces of Work

To solve the CSP, we first split the vehicle itineraries originated from an MDVSP solution into pieces of work. To exemplify this operation, assume that:

- All start and end trip points are also relief points. That is, each vehicle trip is a task;
- The minimum and maximum durations of a piece of work are, respectively, 30 minutes and 5 hours.

Table 3.4 shows a vehicle's itinerary split into three pieces of work. On the left is the vehicle itinerary and, on the right, the pieces of work formed (*Piece 1*, *Piece 2*, *Piece 3*).

A crew is responsible for each activity of a vehicle outside the depot. This responsibility includes:

1. the trips themselves attributed to the vehicle,
2. the moves for positioning the vehicle in the appropriate places (i.e., the so-called deadheads),
3. the waiting time at the stations to wait for the next trip to start.

Algorithm 6: *perturbation*

Data: $S^{vc} = [s^v, S^c]$, a solution for the MDVCSP. s^v is the solution for the MDVSP and $S^c = \{s_1^c, s_2^c, ..., s_n^c\}$ is the set of solutions for the n CSPs (for a problem with n depots).
Data: l, the level of perturbation.
Result: $S_2^{vc} = [s_2^v, S_2^c]$, the solution perturbed.
$s_2^v \leftarrow perturb(s^v, l)$
Let $\{R_1, R_2, ..., R_n\}$, where R_i is the set of pieces of work removed from the i-th depot, that is, the pieces associated with s^v and not associated with s_2^v
Let $\{C_1, C_2, ..., C_n\}$, where C_i is the set of pieces of work created in the i-th depot, that is, the pieces associated with s_2^v and not associated with s^v
$S_2^c \leftarrow \emptyset$ `// the new set of solutions for the` n `CSPs`
for $i \leftarrow 1$ **to** n **do**
    Let $s_i^c \in S^c$
    $s_i^c \leftarrow s_i^c \setminus R_i$
    $piecesOrder(C_i, sortPieces\_pert)$ `// Defines the pieces' order in` C_i
    while $C_i \neq \emptyset$ **do**
        Let p be the first piece of work from C_i
        $C_i \leftarrow C_i \setminus \{p\}$
        $s_i^c \leftarrow s_i^c \cup \{p\}$ `// as described in Subsection 3.3.5.2, Algorithm 9`
    $S_2^c \leftarrow S_2^c \cup \{s_i^c\}$
$S_2^{vc} \leftarrow [s_2^v, S_2^c]$ `// perturbed solution for the MDVCSP`

Algorithm 7: *localSearch*

Data: $S^{vc} = [s^v, S^c]$, a perturbed solution for the MDVCSP. s^v is the solution for the MDVSP and $S^c = \{s_1^c, s_2^c, ..., s_n^c\}$ is the set of solutions for the n CSPs (for a problem with n depots).
Result: $S_2^{vc} = [s_2^v, S_2^c]$, the solution for the MDVCSP after local search.
$S_2^c \leftarrow \emptyset$ `// the new set of solutions for the` n `CSPs`
for $i \leftarrow 1$ **to** n **do**
    Let $s_i^c \in S^c$
    if s_i^c *has been modified* **then**
        $s_i^c \leftarrow VND\_CSP(s_i^c)$ `// as described in Subsection 3.3.5.4, Algorithm 11`
    $S_2^c \leftarrow S_2^c \cup \{s_i^c\}$
$s_2^v \leftarrow s^v$ `// solution for the MDVSP does not change`
$S_2^{vc} \leftarrow [s_2^v, S_2^c]$ `// the solution for the MDVCSP after local search`

Algorithm 8: *acceptanceCriterion*

Data: S_*^{vc}, the current solution for the MDVCSP.
Data: S^{vc}, the solution for the MDVCSP after perturbation and local search.
Data: l, the current level of perturbation of the ILS-MDVCSP.
Data: l_{max}, the maximum level of perturbation.
Result: A pair $[S_*^{vc}, l]$, where S_*^{vc} is the updated current solution for the MDVCSP and l is the updated level of perturbation.

if $f^{vc}(S^{vc}) < f^{vc}(S_*^{vc})$ **then**
    $S_*^{vc} \leftarrow S^{vc}$
    $l \leftarrow 1$
if $f^{vc}(S^{vc}) = f^{vc}(S_*^{vc})$ **then**
    $S_*^{vc} \leftarrow S^{vc}$
    $l \leftarrow l + 1$
if $f^{vc}(S^{vc}) > f^{vc}(S_*^{vc})$ **then**
    $l \leftarrow l + 1$
if $l > l_{max}$ **then**
    $l \leftarrow 1$

Thus, the following attributes are associated with each piece of work:

- Expanded start time: Piece of work start time;
- Expanded start point: Location where the piece of work starts;
- Expanded end time: Piece of work end time;
- Expanded end point: Location where the piece of work ends;
- Duration: Duration of the piece of work.

In Table 3.4, *Piece 1* starts at 7:53 am in depot G1, ends at 11:34 am at point B, and has a total duration of three hours and forty-one minutes. *Piece 2* starts at 11:34 am at point B, ends at 4:14 pm also at point B, and has a total duration of four hours and forty minutes. Therefore, we observe that the *Piece 1* ends precisely at the place and time when the *Piece 2* starts. The same situation occurs between the pieces of work *Piece 2* and *Piece 3*. In this way, the vehicle is never without a crew outside the depot.

According to the piece of work definition (see Section 3.1), there are many ways to partition a vehicle itinerary to build pieces of work. However, we propose only two heuristic procedures to do this. They are: direct partitioning and reverse partitioning. The objective in both procedures is to build pieces of work that have as many trips as possible and do not include waiting times in the depot.

Table 3.4: Example of direct partitioning of the vehicle itinerary into pieces of work

Vehicle Itinerary					Piece of Work					
Trip Number	Start Time	Start Point	End Time	End Point	Exp. Start Time	Exp. Start Point	Exp. End Time	Exp. End Point	Duration	
-	7:53	G1	8:35	A	7:53	G1	11:34	B	3:41	*Piece 1*
1	8:35	A	9:18	D						
-	9:18	D	9:42	A						
2	9:55	A	10:33	D						
-	10:33	D	10:55	B						
3	11:34	B	12:24	C	11:34	B	16:14	B	4:40	*Piece 2*
4	13:31	C	13:56	A						
5	14:01	A	14:26	C						
6	14:30	C	15:23	B						
7	16:14	B	17:07	C	16:14	B	20:04	G1	3:50	*Piece 3*
8	17:31	C	17:56	A						
9	18:01	A	18:26	C						
10	18:30	C	19:23	B						
-	19:23	B	20:04	G1						

Table 3.5: Example of inverse partitioning of the vehicle itinerary into pieces of work

Vehicle Itinerary					Piece of Work					
Trip Number	Start Time	Start Point	End Time	End Point	Exp. Start Time	Exp. Start Point	Exp. End Time	Exp. End Point	Duration	
-	7:53	G1	8:35	A	7:53	G1	10:33	D	2:40	*Piece 1*
1	8:35	A	9:18	D						
-	9:18	D	9:42	A						
2	9:55	A	10:33	D						
-	10:33	D	10:55	B	10:33	D	15:23	B	4:50	*Piece 2*
3	11:34	B	12:24	C						
4	13:31	C	13:56	A						
5	14:01	A	14:26	C						
6	14:30	C	15:23	B						
7	16:14	B	17:07	C	15:23	B	20:04	G1	4:41	*Piece 3*
8	17:31	C	17:56	A						
9	18:01	A	18:26	C						
10	18:30	C	19:23	B						
-	19:23	B	20:04	G1						

In direct partitioning, the procedure starts in the depot, before the vehicle's first trip on the day. The procedure systematically covers the entire vehicle itinerary, trip by trip, and forms pieces of work. In this approach, a piece of work always:

- starts in the depot or at the associated first trip beginning;
- ends in the depot or at the first trip beginning of the next piece of work.

Table 3.4 illustrates an example of the generation of pieces of work by direct partitioning.

In the reverse partitioning procedure, the pieces of work' construction start in the depot after the vehicle's last trip. The method systematically goes through the entire

vehicle itinerary, from back to front, building the pieces of work. In this case, a piece of work always:

- starts in the depot or at the end of the last trip of the previous piece of work;
- ends in the depot or at the associated last trip ending.

See an example of this type of partitioning in Table 3.5.

We create pieces of work aiming to reduce the complexity of solving the CSP in two ways:

1. Combining tasks to form pieces of work. Thus, the piece of work becomes the smallest amount of work that we can assign to a crew. Furthermore, we impose that each duty is composed of at most two pieces of work;
2. Partitioning the vehicle itineraries into pieces of work using only two heuristic procedures, i. e., we do not consider all possible splits.

3.3.5.2 Heuristic Method for Inserting a Piece of Work Into Crew Schedule

Let be the following data:

- p: piece of work to be included in the schedule;
- s_0^c: solution partially built for the CSP and that always keeps an empty duty in the last position on its list of duties (see representation in Subsection 3.3.1.2).

We defined three basic methods for inserting p into s_0^c maintaining the feasibility of the solution, they are:

1. Random insertion - $randomInsertion(p, s_0^c)$: We randomly select a crew from the schedule and, if the viability of s_0^c is maintained, we insert the piece of work p into its duty. As there is a possibility of failure, this procedure returns *true* if the insertion is effective and $false$, otherwise;
2. Sequential insertion - $sequentialInsertion(p, s_0^c)$: The list of duties for the s_0^c solution is inspected sequentially, starting from the first position. The piece of work p is assigned to the first crew that can perform it;
3. Greedy insertion - $greedyInsertion(p, s_0^c)$: We analyze the cost of inserting the piece of work p in each of the possible duties of s_0^c. In this way, p is allocated to the lowest cost duty according to the evaluation function (3.14).

Since we keep an empty duty in s_0^c, we will always find a duty to insert p in the sequential and greedy insertion procedures. That is, if there is no other possibility, the empty duty of s_0^c will receive p.

Algorithm 9 shows the heuristic procedure for inserting a piece of work p into the partially built crew schedule s_0^c. This procedure combines the three primary methods presented above (random, sequential, and greedy insertions).

According to Algorithm 9, we initially tried, at most it_{max} times, to randomly insert p in a duty (lines 7 to 9). If we can't, we randomly choose one of the two remaining types of insertions, sequential or greedy, to insert the piece of work p (lines 10 and 11). To define $it_{\max}$ (line 4), we consider the percentage of duties in s_0^c that we can test. This percentage is given by *percDutiesTested* and consists of a parameter whose value we need to specify. Finally, in lines 12 and 13, we guarantee that s_0^c will have an empty duty in the last position in its duty list.

Algorithm 9: *insertPiece_Solution*

1 **Data:** p, a piece of work to insert in the solution.
2 **Data:** s_0^c, a solution partially built for the CSP. It has an empty duty at the last position in its duties list.
3 **Result:** s_1^c, a partially built solution for the CSP that includes the piece of work p. It has an empty duty at the last position in its duties list.
4 $it_{max} \leftarrow percDutiesTested \times$ number of duties in s_0^c `// the maximum number of attempts to insert` p `randomly into` s_0^c
5 $it \leftarrow 0$ `// the number of attempts made to insert` p `randomly into` s_0^c
6 $inserted \leftarrow false$ `// boolean variable that will be true if` p `is inserted randomly into` s_0^c
7 **while** $it < it_{max}$ *and* $inserted = false$ **do**
8 $\quad inserted \leftarrow randomInsertion(p, s_0^c)$
9 $\quad it \leftarrow it + 1$
10 **if** $inserted = false$ **then**
11 $\quad sequentialInsertion(p, s_0^c)$ or $greedyInsertion(p, s_0^c)$, choose randomly between one of these procedures
12 **if** s_0^c *does not have an empty duty* **then**
13 $\quad$ Insert empty duty in the last position in the s_0^c duty list
14 $s_1^c \leftarrow s_0^c$

3.3.5.3 Heuristic Method for Generating an Initial Solution for the CSP

Algorithm 10 describes the heuristic method that generates an initial solution for the CSP. For this procedure, we must supply the set of pieces of work of the associated vehicle schedule. Due to the construction process, in set P, each vehicle's pieces of work are arranged sequentially. However, we can also sort this set by the start time of the pieces of work. The parameter *sortPieces_const* defines the order of the pieces of work in the set P (line 3). Then, we allocate each piece of work to a crew, exactly as presented in Algorithm 9.

Algorithm 10: *constructive_CSP*

1 **Data:** P, the set of pieces of work.
2 **Result:** s^c, a solution for the CSP.
3 $piecesOrder(P, sortPieces\_const)$ `// Defines the pieces' order in` P
4 Start s^c with just one empty duty
5 **while** $P \neq \emptyset$ **do**
6 Let p be the first piece of work from P
7 $P \leftarrow P \setminus \{p\}$
8 $s^c \leftarrow s^c \cup \{p\}$ `// as described in Subsection 3.3.5.2, Algorithm 9`

3.3.5.4 Heuristic Method of Local Search for the CSP

To improve a CSP solution, we propose a local search method based on the B-VND procedure described in Subsection 2.2.1.1.

Algorithm 11 presents the developed heuristic, the VND-CSP. Given the high computational cost of evaluating a whole neighborhood of the current solution at each iteration, we employ a selection at random of an improving neighbor as search strategy. The search methods that we developed are:

1. *relocate_CSP* (Algorithm 12): explores the neighborhood N_r^c (relocation of a piece of work);
2. *exchange_CSP* (Algorithm 13): explores the neighborhood N_e^c (exchange of pieces of work).

Algorithm 11: VND-CSP

```
1  Data: s^c, a solution for the CSP.
2  Result: s^c_*, the solution for the CSP after local search.
3  k ← 1
4  while k ≤ 2 do
5      if k = 1 then
6          s^c_1 ← relocate_CSP(s^c)
7      if k = 2 then
8          s^c_1 ← exchange_CSP(s^c)
9      if f(s^c_1) < f(s^c) then
10         s^c ← s^c_1
11         k ← 1
12     else
13         k ← k + 1
14 s^c_* ← s^c
```

Algorithm 12: *relocate_CSP*

```
1  Data: s^c_0, a solution for the CSP.
2  Result: s^c, the solution for the CSP after search.
3  s^c ← s^c_0
4  for i ← 1 to it_r_max do
5      choose s^c_1 ∈ N^c_r(s^c) at random
6      if f(s^c_1) < f(s^c) then
7          s^c ← s^c_1
8          break
9      else
10         i ← i + 1
```

The *relocate_ CSP* method works as follows. Given a CSP solution s^c, we apply a move to it, generating a neighbor s_1^c. If s_1^c represents an improvement for the current solution's value s^c, then, this neighbor becomes the current solution, and the method ends; otherwise, we randomly generate a new neighbor. If after $it\_r_{\max}$ iterations we do not find an improved solution, then we finish this method. The value of $it\_r_{\max}$ is estimated by Equation (3.16):

$$it\_r_{\max} = \mid N_r^c(s^c) \mid \times perc\_r, \tag{3.16}$$

Algorithm 13: *exchange_CSP*

```
Data: s_0^c, a solution for the CSP.
Result: s^c, the solution for the CSP after search.
s^c ← s_0^c
for i ← 1 to it_e_max do
    choose s_1^c ∈ N_e^c(s^c) at random
    if f(s_1^c) < f(s^c) then
        s^c ← s_1^c
        break
    else
        i ← i + 1
```

where:

(a) $it\_r_{\max}$ is the maximum number of iterations with no improvement in the current solution;

(b) $N_r^c(s^c)$ is the set of neighbors of s^c concerning the neighborhood structure N_r^c;

(c) *perc_ r* is the percentage of the size of the neighborhood $N_r^c(s^c)$ to be explored.

Note that this calculation links $it\_r_{\max}$ to the instance's size to be solved. Thus, as we explore only part of the neighborhood, there is a reduction in its evaluation cost.

The *exchange_ CSP* method works likewise the *relocate_ CSP* method. Furthermore, we estimate the value of $it\_e_{\max}$ similarly to Equation (3.16), replacing *perc_ r* with *perc_ e*.

As the Algorithm 11 shows, we end the VND-CSP method when there is no improvement in the two neighborhood structures, N_r^c and N_e^c. In this case, the method returns a local optimum concerning these two neighborhoods. In the VND-CSP, we apply the *relocate_ CSP* and *exchange_ CSP* procedures in this order.

Chapter 4

Computational Experiments

This chapter reports the computational experiments that deal with the MDVCSP and use the ILS-MDVCSP matheuristic algorithm proposed in Chapter 3. Next, we make some considerations about the execution of these experiments.

The ILS-MDVCSP was developed in the C++ language and compiled with version 9.3.0 of gcc. Subsection 4.1.2 describes the tuning process of the ILS-MDVCSP's parameters. The mathematical models developed for the MDVSP were solved using the standard configuration (except for the number of threads) of the Gurobi solver version 9.0.3. The experiments were performed on an Intel (R) Xeon (R) E5-2640 (2.50 GHz) microcomputer and under the 64-bit GNU Linux Ubuntu 20.04.1 LTS operating system. All experiments were run using a single thread only.

To perform the computational experiments, we use the instances of Huisman et al. [2005], which are widely used in the literature. We also consider real-world instances originating from the public urban bus transport system from one of the largest Brazilian cities (Belo Horizonte/MG). In Sections 4.1 and 4.2, we describe the characteristics of these instances and present and analyze the results obtained by the proposed algorithm. The benchmark and Brazilian real-world instances, executable code of the implemented algorithms, and all scripts to run the executable and solve the instances are available for download at `http://sites.google.com/view/mdvcsp-instances` (generally, it is not accessible from a Google Workspace account).

The cost values used in the evaluation functions presented for the MDVSP (3.13), CSP (3.14) and MDVCSP (3.15) are the same ones used in Steinzen et al. [2010]. Table 4.1 shows these costs. According to it, there is a fixed cost of 1000 units for each vehicle and crew, a variable cost of 1 unit for each minute a vehicle is outside the depot, and a variable cost of 0.1 unit for each minute crew working time. So, these costs prioritize the minimization of the number of vehicles and crews employed. The reduction of operating costs (variable costs) is a secondary objective.

Table 4.1: Costs considered in the evaluation functions and their respective values

Type of cost	Value
vehicleCost	1000
crewCost	1000
operationalCost	1
workingTimeCost	0.1

4.1 Literature Instances

In this section, we describe the instances of the literature in Subsection 4.1.1 and the tuning process of our algorithm in Subsection 4.1.2. We compare the results of our algorithm with those of other methods from the literature in Subsection 4.1.3.

4.1.1 Instances Description

We use eight groups of instances that depict characteristics of European public transport companies. They are differentiated by the number of trips, which can be: 80, 100, 160, 200, 320, 400, 640, or 800 trips. Each group contains ten instances, totaling 80 instances. The first six groups of instances were made available in Huisman [2003], and the last two in Steinzen [2007a]. Huisman and Steinzen generated all of these instances from the software described in Huisman [2004] and Huisman et al. [2005]. These instances are widely used in the literature (see Table 2.1 in Section 2.1) and have the following characteristics:

- There are four depots;
- Depots have unlimited capacity. That is, the number of vehicles and crews available for each depot is unlimited;
- Each timetable trip can be serviced by any depot;
- Instances with 80, 160, and 320 trips have four *relief points*, and the others have five.

Additionally, according to Huisman et al. [2005], whenever a duty starts or ends its activity in the depot, a crew preparation time of 10 and 5 minutes is required, respectively. However, a duty can start or end at any other relief point (outside the depot), and, in

this case, the crew preparation time will be the travel time between the relief point and the depot plus 15 minutes.

Still according to Huisman et al. [2005], there are five types of duties: *tripper*, formed by only one piece of work lasting between 30 minutes and 5 hours; and four more types (*early*, *day*, *late*, and *split*), made up of two pieces of work. Table 4.2 details them, whose characteristics are as follows:

- Interval: Interval of the day on which a duty can take place. The absence of a lower or upper limit indicates that there is no restriction on the time of beginning or end of the duty, respectively;
- Piece length: Minimum and maximum length allowed for a piece of work. It is noteworthy that during a piece of work, the crew will remain responsible for the same vehicle without interruption;
- Break length: Minimum break duration between the pieces of work of the duty;
- Duty length: Maximum duration of the duty considering all activities: preparation to start or end the duty, monitoring of the vehicle and mandatory breaks;
- Working time: Maximum time a crew can spend on the vehicle. It is the sum of the duration of the duty' pieces of work.

Table 4.2: Characteristics of the different duty types

	Early		*Day*		*Late*		*Split*	
	Min	Max	Min	Max	Min	Max	Min	Max
Interval		16:30	8:00	18:14	13:15			19:30
Piece length	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00
Break length	0:45		0:45		0:45		1:30	
Duty length		9:45		9:45		9:45		12:00
Working time		9:00		9:00		9:00		9:00

4.1.2 Parameter Settings

For tuning the developed ILS-MDVCSP algorithm's parameters, we apply the irace package [López-Ibáñez et al., 2016b]. This software tunes the parameters of optimization algorithms. The authors developed it in the R language and implemented an extension of the Iterated F-race algorithm (I / F-Race) [Birattari et al., 2010].

Irace receives as input a set of values assumed by the parameters, a set of instances for tuning those parameters, and a set of options for running irace.

In Table 4.3, we present the analyzed parameters, describe their meanings, the tested values, and highlight in bold the values returned by irace.

Table 4.3: Parameters of the proposed algorithm

Parameter	Description	Tested and returned values
percDutiesTested	Percentage of crew members evaluated. It is used in Algorithm 9, described in Subsection 3.3.5.2.	$0.0, \mathbf{0.25}, 0.5, 0.75, 1.0$
sortPieces_const	Defines whether or not the set of pieces of work should be ordered by the pieces' start time. It is used in Algorithm 10 (*constructive_CSP*), presented in Subsection 3.3.5.3.	(a) Yes (b) **No**
sortPieces_pert	Defines whether the set of pieces of work should be ordered by the pieces' start time. It is used in Algorithm 6 (*perturbation*), described in Subsection 3.3.4.	(a) Yes (b) No (c) **Set randomly**
perc_r	Percentage of the size of the neighborhood $N_r^c(s^c)$ to be explored. It is used in (3.16) of Subsection 3.3.5.4.	$0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, \mathbf{0.7}, 0.8, 0.9, 1.0$
perc_e	Percentage of the size of the neighborhood $N_r^c(s^c)$ to be explored. It is defined in Subsection 3.3.5.4.	$0.0, 0.1, \mathbf{0.2}, 0.3, 0.4, 0.5, 0.6, 0.7$
l_{max}	Perturbation levels used in the ILS-MDVCSP matheuristic algorithm. Levels 1 to 6 are described in Subsection 3.3.4.	(a) From 1 to 3 (b) From 1 to 4 (c) From 1 to 5 (d) **From 1 to 6**

Among the eight groups of instances of the literature defined in Subsection 4.1.1, we randomly choose an instance from each group of instances with 80, 200, 400, and 800 trips for the irace tuning phase.

We run irace in its default configuration, except for the option *maxExperiments*, for which we assign the value 250. This option defines the number of times that the ILS-MDVCSP algorithm will be executed during the tuning process.

From statistical tests, irace iteratively generates and tests different parameter configurations for the ILS-MDVCSP algorithm. At the end of its execution, irace returns the elite configuration, that is, the configuration for the parameters that provided the best average performance of the algorithm.

According to the values tested for each parameter in Table 4.3, there are more than 10500 possible parameter configurations. Therefore, the manual tuning of the ILS-MDVCSP algorithm's parameters would be highly costly and possibly inefficient, which justifies the use of irace.

For more details on the irace package, we recommend the user's guide López-Ibáñez et al. [2016a].

4.1.3 Results

All 80 instances were solved ten times by the ILS-MDVCSP algorithm. We fixed the run time of ILS-MDVCSP at 15 minutes for small instances (with 80, 100, 160, and 200 trips) and 40 minutes for the large ones (with 320, 400, 640, and 800 trips).

Table 4.4 reports the best results found by the proposed algorithm, comparing them with those of the literature methods, namely: HK_19 [Horváth and Kis, 2019], KAA_12 [Kliewer et al., 2012], SGSK_10 [Steinzen et al., 2010], BLW_08 [Borndörfer et al., 2008], and HFW_05 [Huisman et al., 2005]. These papers deal with the MDVCSP as proposed in Huisman et al. [2005].

In Table 4.4, *cpu* is the average time, in minutes, reported for solving each group of instances. In line *cpu adj.*, we adjust this value to match the machine where the tests were performed with the machine described in Borndörfer et al. [2008], which has the most simple configuration machine considered in this comparison. For this match, we use the cpu benchmark website PassMark Software Pty Ltd [1998], which provides benchmark results for CPUs for more than 600,000 systems, covering more than 1200 different types of CPUs. This adjustment allows for a fair comparison of the run time of the approaches.

Lines *vehicles*, *crews*, and $v + c$ of Table 4.4 report the average number of vehicles, crews, and sum of vehicles and crews for each group of instances, respectively. This table does not report the following characteristics used to evaluate a solution ((3.13) and (3.14)): vehicle operating time, the crew working time, and cost. We did so because Kliewer et al. [2012], Steinzen et al. [2010], Borndörfer et al. [2008], and Huisman et al. [2005] did not provide this information.

We use the Relative Percentage Deviation (RPD_i^{Alg}) to evaluate the average sum of vehicles and crews, $v + c$, generated by each method Alg for the group of instances i. It is calculated according to (4.1):

$$RPD_i^{Alg} = \frac{(v+c)_i^{Alg} - (v+c)_i^{best}}{(v+c)_i^{best}}, \tag{4.1}$$

where $(v + c)_i^{Alg}$ is the $v + c$ obtained by method Alg for the group of instances i and $(v+c)_i^{best}$ is the best known $v+c$ for the group of instances i. Then, the RPD_i^{Alg} informs the deviation percentage of the $v + c$ found by method Alg concerning the best known $v + c$ for the group of instances i.

In Table 4.4, we present the RPD for each method and group of instances that we are considering. Moreover, the best results for each evaluation criterion (number of vehicles, crews, vehicles plus crews, and RPD) are highlighted in bold. Note that the methods in the literature did not handle some large instances. In these cases, no information appears in Table 4.4. Besides, Huisman et al. [2005] did not report the run time of their algorithm and Kliewer et al. [2012] did not detail the average number of vehicles and crews separately for the group of instances with 640 trips.

Concerning the average sum of vehicles and crews $v+c$ of Table 4.4, we can see that the proposed algorithm gives the smallest values for six of the eight analyzed groups of instances. Furthermore, our algorithm is the only one to find the best results on instance groups with 200, 640, and 800 trips. It only loses to HK_19, KAA_12, and SGSK_10 in small instances, involving 80 and 100 trips.

Regarding the average number of vehicles in Table 4.4, the ILS-MDVCSP algorithm was able to obtain the smallest values for all the groups of instances. So, after the MDVSP solving in optimality by the proposed algorithm, the improvement step of the solution for MDVCSP has not compromised the quality of the vehicle scheduling.

The HK_19 method is the only one that obtains the exact solution to the problem. However, this approach was able to solve only small instances, with 80 or 100 trips. Of the 20 instances considered, they found the optimal solutions in four instances, and, for seven instances, the lower limit gap was less than 0.5%. As shown in Table 4.4, they tested instances with 160 trips, but the procedure for generating columns at the root node consumed alone more than three hours on average. Thus, for large instances, the solution process is very time-consuming and fails to generate better quality solutions than those found in the present work.

Regarding the adjusted run time of Table 4.4, as the instance's size increases, our approach becomes substantially less costly than the others presented in the literature. For the largest group of instances treated by KAA_12 and SGSK_10, with 640 trips, the ILS-MDVCSP algorithm performed best with less than 10% of the processing time they used. Besides, our algorithm was the only one to treat the group of instances with 800 trips. These observations show the ability of the proposed algorithm to handle large instances satisfactorily.

In Table 4.5, we analyze the variability of the solutions obtained by the ILS-MDVCSP algorithm concerning the sum of vehicles and crews $(v + c)$. In this table, *best* and *average* refer, respectively, to the best and average values of $v + c$ per group of instances and in ten runs. The RPD_i^{avg} informs the deviation percentage of the average of $v + c$ concerning the best value of $v + c$ for the group of instances i. It is calculated according to Equation (4.2):

$$RPD_i^{avg} = \frac{(v+c)_i^{avg} - (v+c)_i^{best}}{(v+c)_i^{best}}, \tag{4.2}$$

Table 4.4: Results from literature instances

Approach	Group of instances							
	80	100	160	200	320	400	640	800
ILS-MDVCSP [1]								
cpu	15.00	15.00	15.00	15.00	40.00	40.00	40.00	40.00
cpu adj.	33.45	33.45	33.45	33.45	89.20	89.20	89.20	89.20
vehicles	**9.20**	**11.00**	**14.80**	**18.40**	**26.70**	**32.90**	**56.90**	**66.90**
crews	19.70	23.10	**31.70**	**38.50**	**55.80**	**67.90**	**119.40**	**142.20**
v + c	28.90	34.10	**46.50**	**56.90**	**82.50**	**100.80**	**176.30**	**209.10**
RPD (%)	3.21	2.40	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
HK_19 [2]								
cpu	15.24	16.78	>> 180.00	-	-	-	-	-
cpu adj.	32.77	36.08	>> 387.00	-	-	-	-	-
vehicles	9.50	11.40	-	-	-	-	-	-
crews	**18.50**	**21.90**	-	-	-	-	-	-
v + c	**28.00**	**33.30**	50.50	-	-	-	-	-
RPD (%)	**0.00**	**0.00**	8.60	-	-	-	-	-
KAA_12 [3]								
cpu	5.43	8.72	22.80	30.40	158.85	183.63	455.60	-
cpu adj.	11.13	17.88	46.74	62.32	325.64	376.44	933.98	-
vehicles	**9.20**	**11.00**	**14.80**	**18.40**	**26.70**	**32.90**	-	-
crews	19.20	22.80	**31.70**	39.00	56.40	69.50	-	-
v + c	28.40	33.80	**46.50**	57.40	83.10	102.40	178.50	-
RPD (%)	1.43	1.50	**0.00**	0.88	0.73	1.59	1.25	-
SGSK_10 [4]								
cpu	3.92	6.15	26.32	45.50	238.75	338.67	953.92	-
cpu adj.	4.31	6.77	28.95	50.05	262.63	372.53	1049.31	-
vehicles	**9.20**	**11.00**	**14.80**	**18.40**	**26.70**	**32.90**	**56.90**	-
crews	19.10	22.70	31.80	38.80	**55.80**	**67.90**	120.40	-
v + c	28.30	33.70	46.60	57.20	**82.50**	**100.80**	177.30	-
RPD (%)	1.07	1.20	0.22	0.53	**0.00**	**0.00**	0.57	-
BLW_08 [5]								
cpu	13.00	21.00	44.00	106.00	328.00	720.00	-	-
cpu adj.	13.00	21.00	44.00	106.00	328.00	720.00	-	-
vehicles	**9.20**	11.20	15.00	18.50	**26.70**	33.10	-	-
crews	20.40	24.50	32.70	40.50	56.10	68.90	-	-
v + c	29.60	35.70	47.70	59.00	82.80	102.00	-	-
RPD (%)	5.71	7.21	2.58	3.69	0.36	1.19	-	-
HFW_05								
cpu	-	-	-	-	-	-	-	-
cpu adj.	-	-	-	-	-	-	-	-
vehicles	**9.20**	**11.00**	**14.80**	**18.40**	-	-	-	-
crews	20.50	25.30	34.10	41.60	-	-	-	-
v + c	29.70	36.30	48.90	60.00	-	-	-	-
RPD (%)	6.07	9.01	5.16	5.45	-	-	-	-

[1] Intel Xeon E5-2640 2.50 GHz/4 GB using only one single thread.
[2] Intel Xeon X5650 2.67 GHz/4 GB using only one single thread.
[3] Dell OptiPlex 755, Intel Core 2 Duo 3.0 GHz/4 GB using only one single thread.
[4] Dell OptiPlex GX620, Intel Pentium IV 3.4 GHz/2 GB using only one single thread.
[5] Dell Precision 650, Intel Dual Xeon 3.0 GHz/4 GB using only one single thread.

where $(v+c)_i^{avg}$ is the average of $v+c$ and $(v+c)_i^{best}$ is the best value of $v+c$ in ten runs for the group of instances i.

Table 4.5: Variability of the solutions obtained by the ILS-MDVCSP algorithm

$v+c$	Group of instances							
	80	100	160	200	320	400	640	800
best	28.90	34.10	46.50	56.90	82.50	100.80	176.30	209.10
average	29.40	34.79	47.32	57.84	83.84	102.05	178.28	210.90
RPD_i^{avg} (%)	1.73	2.02	1.76	1.65	1.62	1.24	1.12	0.86

Table 4.5 shows that the deviation between the average and best value of $v+c$ for each group of instances is slight. Note that the values of RPD_i^{avg} vary from 0.86% to 2.02% only.

4.2 Belo Horizonte Instances

In this section, we describe the instances of Belo Horizonte in Subsection 4.2.1. In Subsection 4.2.2, we compare the companies' solutions and the results of our algorithm in two versions, ILS-MDVCSP and ILS-SDVCSP. The ILS-SDVCSP solves the VCSP for each depot separately.

4.2.1 Instances Description

The real-world instances considered in this work come from a given region of Belo Horizonte/MG, Brazil. In these instances, there are four depots (D01, D02, D03, and D04), which operate on four days of the week with different timetables (Monday, Friday, Saturday, and Sunday). Timetables from Tuesday to Thursday are the same as Monday.

The companies studied firstly assign trips to depots. Then, they generate the vehicle and crew schedules sequentially, considering one depot at a time. In Table 4.6, instances 1 to 16 were provided by some companies. The other instances were created by us from the original ones and considered two, three, or four depots together. In all instances, each depot contains a limited fleet of identical vehicles. Table 4.6 shows the

characteristics of these instances. For each instance, we report the number of trips, the total time of trips (in the format hours : minutes), the number of relief points, the size of the available fleet currently in each depot, and the number of depots considered.

Table 4.6: Characteristics of the Belo Horizonte instances

#id	Instances	Number of trips	Time of trips (h:m)	Number of relief points	Fleet size	Number of depots
1	D01_MON	260	443:19	3	41	1
2	D01_FRI	260	453:39			
3	D01_SAT	172	270:40			
4	D01_SUN	90	158:33			
5	D02_MON	468	527:02	3	35	1
6	D02_FRI	468	524:27			
7	D02_SAT	359	388:32			
8	D02_SUN	298	315:00			
9	D03_MON	206	406:32	2	29	1
10	D03_FRI	203	407:43			
11	D03_SAT	130	255:20			
12	D03_SUN	108	218:02			
13	D04_MON	639	844:57	3	64	1
14	D04_FRI	639	856:32			
15	D04_SAT	441	552:42			
16	D04_SUN	332	434:43			
17	D01-D02_MON	728	970:21	4	D01: 41 D02: 35	2
18	D01-D02_FRI	728	978:06			
19	D01-D02_SAT	531	659:02			
20	D01-D02_SUN	388	473:33			
21	D01-D02-D03_MON	934	1376:53	5	D01: 41 D02: 35 D03: 29	3
22	D01-D02-D03_FRI	931	1385:49			
23	D01-D02-D03_SAT	661	914:21			
24	D01-D02-D03_SUN	496	691:35			
25	D01-D02-D03-D04_MON	1573	2221:50	6	D01: 41 D02: 35 D03: 29 D04: 64	4
26	D01-D02-D03-D04_FRI	1570	2242:21			
27	D01-D02-D03-D04_SAT	1102	1467:03			
28	D01-D02-D03-D04_SUN	828	1126:18			

4.2.2 Results

The hybrid algorithm ILS-MDVCSP developed for the MDVCSP was adapted to handle the SDVCSP (single-depot vehicle and crew scheduling problem) (ILS-SDVCSP). To this end, we use the same mathematical model developed for the MDVSP to solve the SDVSP and consider only the ILS-MDVCSP perturbation levels 2 to 6, described in Subsection 3.3.4. We prevented the depot exchange.

We ran the ILS-SDVCSP and ILS-MDVCSP algorithms ten times for each instance. The results reported below refer to the averages obtained. For the tests related to two and three depots, we set the time t for each algorithm to run at $t = d$ hours, with d being the number of depots of the instance. For the tests considering four depots, we set $t = 3.75d$ hours to solve the instances of the Monday and Friday and $t = 1.25d$ hours to solve the instances of the Saturday and Sunday.

Tables 4.7, 4.8, and 4.9 show the features of the solutions obtained for, respectively, the D01 and D02 depots; D01, D02, and D03 depots; and D01, D02, D03, and D04 depots. For the company and the ILS-SDVCSP algorithm, the data correspond to the union of the solutions obtained for each depot solved separately (instances 1 to 16 in Table 4.6). That is, only the ILS-MDVCSP solved the instances with, respectively, two, three, and four depots simultaneously (instances from 17 to 28 in Table 4.6).

The attributes in Tables 4.7, 4.8, and 4.9 that we have not yet described are: *vehicle time* – the total operating time of the vehicles (in minutes), *crew time* – the working time considering all crews (in minutes), and *total cost* – which represents the cost of the solution calculated as specified in Subsection 3.3.3. We also show separately the number of vehicles (v) and crews (c) used by each depot (i.e., *Di: v / c* for the i-th depot).

As the companies did not provide their crew schedules, we did not report this information. Moreover, we consider the same types of duties proposed by Huisman et al. [2005] for defining the crew scheduling in the ILS-SDVCSP and ILS-MDVCSP algorithms.

According to Tables 4.7, 4.8, and 4.9, for all instances, the ILS-SDVCSP algorithm solutions are better than those used by the companies considering the features *vehicles* and *vehicle time*. When we analyze the depots D01 and D02 together (Table 4.7), the ILS-SDVCSP saved on average 5.75 vehicles per day and reduced on average by about 11% the daily operating time of the vehicles (considering the four days of the week with different timetables). Analyzing the depots D01, D02, and D03 together (Table 4.8), the ILS-SDVCSP saved on average 8.5 vehicles per day and reduced on average by about 10% the daily operating time of the vehicles (considering the four days of the week with different timetables). Examining the depots D01, D02, D03, and D04 together (Table 4.9), the ILS-SDVCSP saved on average 19 vehicles per day and reduced on average by about 11% the daily operating time of the vehicles (considering the four days of the week with different timetables). This result shows that although the companies did not provide the crew schedules, their fleet's higher operational time indicates the need for more labor time (crews) concerning the solutions from the ILS-SDVCSP (see constraint C8 of Section 3.1).

Tables 4.7, 4.8, and 4.9 also show that the ILS-MDVCSP was the approach that obtained the best results. Its solutions are of higher quality than those of the companies and ILS-SDVCSP for all evaluation criteria. Regarding the companies' solutions, when we analyze the depots D01 and D02 together (Table 4.7), the ILS-MDVCSP saved on average

8.25 vehicles per day and reduced on average by about 14% the daily operating time of the vehicles (considering the four days of the week with different timetables). Examining the depots D01, D02, and D03 together (Table 4.8), the ILS-MDVCSP saved on average 12.25 vehicles per day and reduced on average by about 15% the daily operating time of the vehicles (considering the four days of the week with different timetables). Analyzing the depots D01, D02, D03, and D04 together (Table 4.9), the ILS-MDVCSP saved on average 25.25 vehicles per day and reduced on average by about 16% the daily operating time of the vehicles (considering the four days of the week with different timetables). Comparing the solutions of the ILS-SDVCSP and ILS-MDVCSP algorithms, we observed that ILS-MDVCSP generated better vehicle and crew schedules for all instances. Thus, we show the relevance of considering more than one depot simultaneously, as already pointed out by the literature, and the potential of the matheuristic proposed in this work, particularly in the context of real-world and large-scale problems.

We also note that, for the instances with fewer trips, referring to Saturday and Sunday, it is possible to completely avoid using the depots without violating the other depots' capacities. In this way, a reduction in these depots' operating costs is allowed on these two days of the week.

As described in Subsection 3.3.4, the ILS-MDVCSP algorithm generates an initial solution for the MDVCSP addressing MDVSP and CSP sequentially. It solves the MDVSP in optimality and solves each CSP with a constructive procedure and a B-VND local search heuristic. From this initial complete solution, the ILS-MDVCSP iteratively approaches the integrated problem. Thus, Table 4.10 reports, for each MDVCSP instance, the initial and final solutions average cost obtained by the ILS-MDVCSP algorithm and the average improvement achieved in percentage. Furthermore, Table 4.10 shows the average cost variations that occurred separately in the solutions of the MDVSP and CSP between the initial and final solutions of the MDVCSP.

From Table 4.10, we can see that the ILS-MDVCSP improved its initial solutions considerably. The improvements range from 9.4% for D01-D02-D03_SAT instance to 12.4% for D01-D02-D03-D04_SUN instance. We also notice that vehicle scheduling suffers a slight cost increase in favor of a relevant reduction in the crew scheduling cost for all instances. In this sense, it is worth remembering that the initial solution of the MDVSP is optimal. Furthermore, a characteristic favorable to a solution for the MDVSP does not always reflect satisfactorily in the CSP and vice versa. Thus, the results show the relevance of the integrated resolution of these problems.

Table 4.11 provides the number of nodes, arcs, and network layers of the time-space network of each Brazilian instance of the VSP. This table also reports the branch-and-bound method's average run time, in seconds and format hours : minutes (hh:mm). Subsection 3.2.2 describes the network structure and mathematical formulation for the MDVSP. See that the number of arcs of the underlying vehicle scheduling network deter-

Table 4.7: Results from the Belo Horizonte instances (2 depots)

Day	Feature	Depots D01 and D02		
		Company	ILS-SDVCSP	ILS-MDVCSP
MONDAY	vehicles	75.0	70.0	67.0
	crews	-	139.9	131.9
	v + c	-	209.9	198.9
	vehicle time	58,221.0	51,276.3	49,997.7
	crew time	-	62,984.9	61,714.0
	D01: v / c	41.0 / -	40.0 / 71.8	32.0 / 65.6
	D02: v / c	34.0 / -	30.0 / 68.1	35.0 / 66.3
	vehicle cost	133,221.0	121,276.3	116,997.7
	crew cost	-	146,198.6	138,071.4
	total cost	-	267,474.9	255,069.1
FRIDAY	vehicles	75.0	70.0	67.0
	crews	-	144.4	135.9
	v + c	-	214.4	202.9
	vehicle time	58,686.0	52,923.6	51,660.0
	crew time	-	64,717.9	63,897.0
	D01: v / c	40.0 / -	39.0 / 74.4	32.0 / 69.7
	D02: v / c	35.0 / -	31.0 / 70.0	35.0 / 66.2
	vehicle cost	187,149.0	122,923.6	118,660.0
	crew cost		150,871.9	142,289.7
	total cost	-	273,795.5	260,949.7
SATURDAY	vehicles	50.0	43.0	40.0
	crews	-	89.2	81.4
	v + c	-	132.2	121.4
	vehicle time	39,552.0	35,421.5	34,065.3
	crew time	-	44,127.7	42,146.1
	D01: v / c	23.0 / -	23.0 / 39.1	5.1 / 8.5
	D02: v / c	27.0 / -	20.0 / 50.1	34.9 / 72.9
	vehicle cost	89,552.0	78,421.5	74,065.3
	crew cost		93,612.9	85,614.7
	total cost	-	172,034.4	159,680.0
SUNDAY	vehicles	32.0	26.0	25.0
	crews	-	63.8	58.5
	v + c	-	89.8	83.5
	vehicle time	28,413.0	24,886.5	23,982.2
	crew time	-	31,602.3	29,595.5
	D01: v / c	11.0 / -	10.0 / 22.5	0.0 / 0.0
	D02: v / c	21.0 / -	16.0 / 41.3	25.0 / 58.5
	vehicle cost	60,413.0	50,886.5	48,982.2
	crew cost		66,960.3	61,459.6
	total cost	-	117,846.8	110,441.8

mines the number of flow variables and most of the constraints.

The numbers of nodes, arcs, and network layers determine the size of an instance in the time-space network representation. Table 4.11 shows that the size of each MDVSP instance increases considerably and in a non-linear manner concerning the sizes of the single-depot instances associated.

Lastly, in Table 4.11, we observed that the time for exact MDVSP resolution increased much for instances with four depots together. The branch-and-bound spent

Table 4.8: Results from the Belo Horizonte instances (3 depots)

Day	Feature	Depots D01, D02, and D03		
		Company	ILS-SDVCSP	ILS-MDVCSP
MONDAY	vehicles	104.0	96.0	92.0
	crews	-	199.5	187.8
	v + c	-	295.5	279.8
	vehicle time	82,613.0	73,986.1	70,937.8
	crew time	-	91,991.7	88,322.0
	D01: v / c	41.0 / -	40.0 / 71.8	41.0 / 84.8
	D02: v / c	34.0 / -	30.0 / 68.1	35.0 / 66.8
	D03: v / c	29.0 / -	26.0 / 59.6	16.0 / 36.2
	vehicle cost	186,613.0	169,986.1	162,937.8
	crew cost	-	208,699.3	196,632.2
	total cost	-	378,685.4	359,570.0
FRIDAY	vehicles	104.0	98.0	94.0
	crews	-	202.8	189.3
	v + c	-	300.8	283.3
	vehicle time	83,149.0	75,693.1	72,626.6
	crew time	-	94,103.0	90,639.0
	D01: v / c	40.0 / -	39.0 / 74.4	41.0 / 84.3
	D02: v / c	35.0 / -	31.0 / 70.0	35.0 / 67.1
	D03: v / c	29.0 / -	28.0 / 58.4	18.0 / 37.9
	vehicle cost	187,149.0	173,693.1	166,626.6
	crew cost		212,210.4	198,363.9
	total cost	-	385,903.5	364,990.5
SATURDAY	vehicles	66.0	57.0	51.0
	crews	-	123.3	108.2
	v + c	-	180.3	159.2
	vehicle time	54,872.0	49,454.3	45,807.7
	crew time	-	63,006.0	56,395.0
	D01: v / c	23.0 / -	23.0 / 39.1	16.0 / 34.7
	D02: v / c	27.0 / -	20.0 / 50.1	35.0 / 73.5
	D03: v / c	16.0 / -	14.0 / 34.1	0.0 / 0.0
	vehicle cost	120,872.0	106,454.3	96,807.7
	crew cost		129,600.7	113,839.5
	total cost	-	236,055.0	210,647.2
SUNDAY	vehicles	50.0	39.0	38.0
	crews	-	92.8	82.6
	v + c	-	131.8	120.6
	vehicle time	41,495.0	36,158.3	33,578.9
	crew time	-	46,567.3	41,228.7
	D01: v / c	11.0 / -	10.0 / 22.5	3.1 / 7.5
	D02: v / c	21.0 / -	16.0 / 41.3	34.9 / 75.1
	D03: v / c	18.0 / -	13.0 / 29.0	0.0 / 0.0
	vehicle cost	91,495.0	75,158.3	71,578.9
	crew cost		97,456.8	86,722.9
	total cost	-	172,615.1	158,301.8

by more than ten hours resolving the largest instances (D01-D02-D03-D04_MON and D01-D02-D03-D04_FRI). Nevertheless, the run time of the ILS-MDVCSP (15 hours) was sufficient to solve the MDVCSP and, simultaneously, satisfactorily meet the practical context of the operational planning.

Table 4.9: Results from the Belo Horizonte instances (4 depots)

Day	Feature	Depots D01, D02, D03, and D04		
		Company	ILS-SDVCSP	ILS-MDVCSP
MONDAY	vehicles	167.0	151.0	146.0
	crews	-	311.0	297.1
	v + c	-	462.0	443.1
	vehicle time	133,310.0	117,408.1	111,834.0
	crew time	-	143,836.5	136,411.0
	D01: v / c	41.0 / -	40.0 / 71.3	41.0 / 89.1
	D02: v / c	34.0 / -	30.0 / 66.2	35.0 / 72.2
	D03: v / c	29.0 / -	26.0 / 59.4	6.0 / 12.8
	D04: v / c	63.0 / -	55.0 / 114.1	64.0 / 123.0
	vehicle cost	300,310.0	268,408.1	257,834.0
	crew cost	-	325,383.8	310,741.1
	total cost	-	593,791.7	568,575.1
FRIDAY	vehicles	168.0	157.0	149.0
	crews	-	320.5	308.1
	v + c	-	477.5	457.1
	vehicle time	134,541.0	121,295.9	116,345.8
	crew time	-	149,196.6	143,994.0
	D01: v / c	40.0 / -	39.0 / 74.1	41.0 / 93.1
	D02: v / c	35.0 / -	31.0 / 68.2	35.0 / 70.0
	D03: v / c	29.0 / -	28.0 / 58.5	9.0 / 17.7
	D04: v / c	64.0 / -	59.0 / 119.7	64.0 / 127.3
	vehicle cost	302,541.0	278,295.9	265,345.8
	crew cost		335,419.8	322,499.4
	total cost	-	613,715.5	587,845.2
SATURDAY	vehicles	111.0	86.0	77.0
	crews	-	190.2	176.2
	v + c	-	276.2	253.2
	vehicle time	88,034.0	78,437.1	73,340.7
	crew time	-	99,021.3	91,700.0
	D01: v / c	23.0 / -	23.0 / 39.3	16.5 / 42.8
	D02: v / c	27.0 / -	20.0 / 49.6	35.0 / 79.6
	D03: v / c	16.0 / -	14.0 / 34.0	0.0 / 0.0
	D04: v / c	45.0 / -	29.0 / 67.3	25.5 / 53.8
	vehicle cost	199,034.0	164,437.1	150,340.7
	crew cost		200,102.3	185,370.0
	total cost	-	364,539.4	335,710.7
SUNDAY	vehicles	85.0	61.0	58.0
	crews	-	145.9	134.6
	v + c	-	206.9	192.6
	vehicle time	67,578.0	58,530.1	55,172.0
	crew time	-	74,597.3	67,847.0
	D01: v / c	11.0 / -	10.0 / 22.1	0.9 / 1.7
	D02: v / c	21.0 / -	16.0 / 41.0	35.0 / 85.2
	D03: v / c	18.0 / -	13.0 / 29.0	0.0 / 0.0
	D04: v / c	35.0 / -	22.0 / 53.8	22.1 / 47.7
	vehicle cost	152,578.0	119,530.1	113,172.0
	crew cost		153,359.9	141,384.7
	total cost	-	272,889.9	254,556.7

Table 4.10: Improvement of the initial solutions in the ILS-MDVCSP

Instances	Initial solution cost	Vehicle cost variation	Crew cost variation	Final solution cost	Improvement (%)
D01-D02_MON	286,434.6	662.7	-32,028.2	255,069.1	11.0
D01-D02_FRI	295,746.9	734.0	-35,531.2	260,949.7	11.8
D01-D02_SAT	178,493.7	207.3	-19,021.0	159,680.0	10.5
D01-D02_SUN	122,188.7	62.2	-11,809.1	110,441.8	9.6
D01-D02-D03_MON	402,221.1	1,325.8	-43,976.9	359,570.0	10.6
D01-D02-D03_FRI	411,074.3	1,379.6	-47,463.4	364,990.5	11.2
D01-D02-D03_SAT	232,432.7	273.7	-22,059.2	210,647.2	9.4
D01-D02-D03_SUN	175,541.2	181.9	-17,421.3	158,301.8	9.8
D01-D02-D03-D04_MON	627,269.1	2,016.0	-60,710.0	568,575.1	9.4
D01-D02-D03-D04_FRI	645,271.0	2,599.8	-60,025.6	587,845.2	8.9
D01-D02-D03-D04_SAT	377,718.9	858.7	-42,866.9	335,710.7	11.1
D01-D02-D03-D04_SUN	290,620.6	529	-36,592.9	254,556.7	12.4

Table 4.11: Characteristics of the exact approach for the VSP instances

Instances	Nodes	Arcs	depot-operating period (network layers)	B&B cpu	
				seconds	hh:mm
D01_MON	942	1,836	1	0.03	00:00
D01_FRI	936	1,829	1	0.03	00:00
D01_SAT	700	1,322	1	0.03	00:00
D01_SUN	371	696	1	0.10	00:00
D02_MON	13,920	29,410	10	13.23	00:00
D02_FRI	13,840	29,360	10	10.65	00:00
D02_SAT	8,001	16,387	7	2.21	00:00
D02_SUN	8,901	17,658	9	3.76	00:00
D03_MON	704	1,321	1	0.02	00:00
D03_FRI	721	1,329	1	0.02	00:00
D03_SAT	485	874	1	0.02	00:00
D03_SUN	401	724	1	0.01	00:00
D04_MON	8,710	19,010	5	12.64	00:00
D04_FRI	10,302	22,656	6	12.62	00:00
D04_SAT	7,266	15,666	6	3.82	00:00
D04_SUN	5,382	11,544	6	1.83	00:00
D01-D02_MON	36,812	85,116	19	370.21	00:06
D01-D02_FRI	36,495	84,609	19	398.64	00:07
D01-D02_SAT	28,251	61,434	18	295.58	00:05
D01-D02_SUN	21,588	44,970	18	99.14	00:02
D01-D02-D03_MON	64,104	157,660	29	2,759.18	00:46
D01-D02-D03_FRI	65,653	162,293	30	2,619.55	00:44
D01-D02-D03_SAT	57,055	129,975	32	1,577.85	00:26
D01-D02-D03_SUN	44,560	97,088	32	756.25	00:13
D01-D02-D03-D04_MON	139,855	413,414	50	37,590.30	10:27
D01-D02-D03-D04_FRI	143,195	424,165	51	46,611.44	12:57
D01-D02-D03-D04_SAT	116,533	305,763	50	14,606.74	04:03
D01-D02-D03-D04_SUN	93,350	230,977	50	5,148.50	01:26

Chapter 5

Conclusions

This work addressed the MDVCSP. This problem involves public transport companies by medium and large buses, which have more than one depot to manage their resources, i.e., the vehicle fleet and crews. In the MDVCSP solution, we deal with two problems in an integrated manner: the MDVSP and the CSP. That is, we simultaneously define vehicle and crew schedules. The objective is to minimize the costs involved and, at the same time, respect the operational restrictions and work regulations. The MDVCSP is an NP-hard optimization problem, and to solve it, we propose a matheuristic algorithm. Our algorithm, called ILS-MDVCSP, uses the Iterated Local Search framework to combine two methods: a Branch-and-Bound method to solve the MDVSP in optimality and a Variable Neighborhood Descent-based method to treat the associated CSPs.

For the tests, we initially used a set of instances well known in the literature. We compared our approach against the main strategies in the literature that addressed the same problem. Our matheuristic algorithm was able to treat instances with 800 trips, a dimension not yet addressed by the current specialized literature. Besides, it obtained the best results for six groups of instances out of the eight groups considered. The run time of our algorithm was shorter for most groups of instances, and, as the instance's size increased, our approach became substantially less costly as compared to the literature.

We also solved the MDVCSP of a region in the city of Belo Horizonte, MG, Brazil. To address the particularities of this problem, we proposed a mathematical formulation based on a time-space network to represent the MDVSP. Regarding the companies' solutions, our algorithm's solutions were considerably better.

Furthermore, we compared the solutions obtained from two integrated approaches: SDVCSP and MDVCSP. So, we developed two algorithms: ILS-SDVCSP and ILS-MDVCSP. The ILS-MDVCSP outperformed the ILS-SDVCSP for all instances.

Therefore, the experiments showed the effectiveness of our matheuristic algorithm to deal with real-world and large-scale problems. We also verified that solving the MDVSP and CSP problems in an integrated manner reduces costs concerning these problems' sequential resolution. Moreover, by considering more than one depot at the same time, we can further reduce costs.

The MDVCSP is a complex problem and little explored in the literature. In prac-

tice, this problem's efficient resolution can bring high savings to the public bus transport sector.

In future work, we aim to propose heuristic algorithms to tackle the MDVSP. These algorithms will be based on metaheuristics and/or the mathematical formulation we proposed for the MDVSP. Thus, we will be able to compare the solutions obtained by the exact and heuristic resolution of the MDVSP. Suppose a heuristic algorithm finds good solutions for MDVSP in an acceptable time. In that case, we will increase the scalability of ILS-MDVCSP and use this heuristic to solve even larger instances of MDVCSP. We also intend to do tests considering different evaluation functions and cost values for the MDVCSP. In the problems we solved, the main objective was to reduce the number of vehicles and crews. However, depending on the context, companies may need to prioritize other schedule' characteristics. So, we plan to consider different real-world scenarios. Moreover, we aim to compare the performance of our algorithm against other methods based on mathematical programming and metaheuristics. Our goal will be to identify the potentials of different optimization techniques to address MDVCSP and related problems. Finally, we note that public bus transport companies often have to deal with delays and interruptions in schedule due to traffic conditions, mechanical troubles with vehicles, crews' no show, and other unexpected events. These delays and interruptions affect the quality of the service provided, user satisfaction, and the companies' operating costs. Thus, it would be interesting to propose an approach for MDVCSP that dynamically updates the schedule to adapt it to some identified unexpected events in real-time.

Bibliography

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows, theory, algorithms, and applications*. Prentice-Hall, New Jersey.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (2001). Minimum cost flow problem. In Floudas, C. A. and Pardalos, P. M., editors, *Encyclopedia of Optimization*, pages 1382–1392. Springer US, Boston, MA.

Amaral, H. F., Urrutia, S., and Hvattum, L. M. (2021). Delayed improvement local search. *Journal of Heuristics*, 27:923–950.

Amberg, B., Amberg, B., and Kliewer, N. (2019). Robust efficiency in urban public transportation: Minimizing delay propagation in cost-efficient bus and driver schedules. *Transportation Science*, 53(1):89–112.

Andrade-Michel, A., Ríos-Solís, Y. A., and Boyer, V. (2021). Vehicle and reliable driver scheduling for public bus transportation systems. *Transportation Research Part B: Methodological*, 145:290–301.

Balas, E. and Padberg, M. W. (1975). Set partitioning. In Roy, B., editor, *Combinatorial Programming: Methods and Applications*, pages 205–258, Dordrecht. Springer Netherlands.

Ball, M. O., Bodin, L. D., and Dial, R. (1983). A matching based heuristic for scheduling mass transit crew and vehicles. *Transportation Science*, 1(17):4–31.

Bertossi, A. A., Carraresi, P., and Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks*, 17(3):271–281.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-race and iterated f-race: An overview. In Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer Berlin Heidelberg, Berlin, Heidelberg.

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151.

Booler, J. M. P. (1975). A method for solving crew scheduling problems. *Journal of the Operational Research Society*, 26:55–62.

Borndörfer, R., Löbel, A., and Weider, S. (2004). A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. *Technical Report ZR 04-14*. Konrad-Zuse Zentrum fuer Informationstechnik, Berlin, Germany.

Borndörfer, R., Löbel, A., and Weider, S. (2008). A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. In Hickman, M., Mirchandani, P., and Voß, S., editors, *Computer-aided Systems in Public Transport*, pages 3–24, Berlin, Heidelberg. Springer Berlin Heidelberg.

Borndörfer, R., Schulz, C., Seidl, S., and Weider, S. (2017). Integration of duty scheduling and rostering to increase driver satisfaction. *Public Transport*, 9(1):177–191.

Boyer, V., Ibarra-Rojas, O. J., and Ríos-Solís, Y. Á. (2018). Vehicle and crew scheduling for flexible bus transportation systems. *Transportation Research Part B: Methodological*, 112:216–229.

Carosi, S., Frangioni, A., Galli, L., Girardi, L., and Vallese, G. (2019). A matheuristic for integrated timetabling and vehicle scheduling. *Transportation Research Part B: Methodological*, 127:99–124.

Ciancio, C., Laganà, D., Musmanno, R., and Santoro, F. (2018). An integrated algorithm for shift scheduling problems for local public transport companies. *Omega*, 75:139–153.

Dauer, A. T. and Prata, B. A. (2021). Variable fixing heuristics for solving multiple depot vehicle scheduling problem with heterogeneous fleet and time windows. *Optimization Letters*, 15:153–170.

Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M., and Soumis, F. (1997). Crew pairing at Air France. *European Journal of Operational Research*, 97(2):245–259.

Desaulniers, G. and Hickman, M. D. (2007). Chapter 2 public transit. In Barnhart, C. and Laporte, G., editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 69 – 127. Elsevier.

Desfontaines, L. and Desaulniers, G. (2018). Multiple depot vehicle scheduling with controlled trip shifting. *Transportation Research Part B: Methodological*, 113:34–53.

Dumitrescu, I. and Stützle, T. (2003). Combinations of local search and exact algorithms. In Cagnoni, S., Johnson, C. G., Cardalda, J. J. R., Marchiori, E., Corne, D. W., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G. R., and Hart, E., editors, *Applications of Evolutionary Computing*, pages 211–223, Berlin, Heidelberg. Springer Berlin Heidelberg.

Elias, S. E. G. (1964). The use of digital computers in the economic scheduling for both man and machine in public transportation. Manhattan, Kansas.

Er-Rbib, S., Desaulniers, G., Hallaoui, I. E., and Bani, A. (2021). Integrated and sequential solution methods for the cyclic bus driver rostering problem. *Journal of the Operational Research Society*, 72(4):764–779.

Fischetti, M., Martello, S., and Toth, P. (1989). The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403.

Freling, R. (1997). *Models and techniques for integrating vehicle and crew scheduling*. PhD thesis, Erasmus University Rotterdam, Amsterdam.

Freling, R., Boender, C. G. E., and Paixão, J. M. P. (1995). An integrated approach to vehicle and crew scheduling. *Technical Report 9503/A*. Econometric Institute, Erasmus University Rotterdam, Rotterdam.

Freling, R., Huisman, D., and Wagelmans, A. P. M. (2003). Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85.

Freling, R., Wagelmans, A. P. M., and Paixão, J. M. P. (1999). An overview of models and techniques for integrating vehicle and crew scheduling. *Computer-Aided Transit Scheduling*, 471:441–460.

Gaffi, A. and Nonato, M. (1999). An integrated approach to ex-urban crew and vehicle scheduling. In *Computer-Aided Transit Scheduling*, pages 103–128. Springer Verlag, Berlin, Germany.

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

Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual.

Haase, K., Desaulniers, G., and Desrosriers, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303.

Hansen, P., Mladenović, N., Brimberg, J., and Pérez, J. A. M. (2019). Variable neighborhood search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 57–97. Springer International Publishing, Cham.

Hansen, P., Mladenović, N., Todosijević, R., and Hanafi, S. (2017). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454.

Horváth, M. and Kis, T. (2019). Computing strong lower and upper bounds for the integrated multiple-depot vehicle and crew scheduling problem with branch-and-price. *Central European Journal of Operations Research*, 27(1):39–67.

Huisman, D. (2003). *Random data instances for multiple-depot vehicle and crew scheduling, accessed Sept. 6, 2019*. `http://people.few.eur.nl/huisman/instances.htm`.

Huisman, D. (2004). Integrated and dynamic vehicle and crew scheduling. Ph.D. thesis, Erasmus University of Rotterdam, The Netherlands.

Huisman, D., Freling, R., and Wagelmans, A. P. M. (2004). A robust solution approach to the dynamic vehicle scheduling problem. *Transportation Science*, 38(4):447–458.

Huisman, D., Freling, R., and Wagelmans, A. P. M. (2005). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502.

Hussain, K., Salleh, M., Cheng, S., and Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, 52:2191–2233.

Ibarra-Rojas, O. J., Delgado, F., Giesen, R., and Muñoz, J. C. (2015). Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75.

Jourdan, L., Basseur, M., and Talbi, E.-G. (2009). Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629.

Kang, L., Chen, S., and Meng, Q. (2019). Bus and driver scheduling with mealtime windows for a single public bus route. *Transportation Research Part C: Emerging Technologies*, 101:145–160.

Kirkman, F. (1968). Problems of innovation in the transport industry: a bus scheduling program. In *Proceedings of PTRC Public Transport Analysis Seminar, Planning and Transport Research and Computation Co. Ltd.*, volume 1, pages 1–15.

Kliewer, N., Amberg, B., and Amberg, B. (2012). Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport*, 3:213–244.

Kliewer, N., Mellouli, T., and Suhl, L. (2006). A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616 – 1627.

Kulkarni, S., Krishnamoorthy, M., Ranade, A., Ernst, A. T., and Patil, R. (2018). A new formulation and a column generation-based heuristic for the multiple depot vehicle scheduling problem. *Transportation Research Part B: Methodological*, 118:457–487.

Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

Laurent, B. and Hao, J. (2008). Simultaneous vehicle and crew scheduling for extra urban transports. In *New Frontiers in Applied Artificial Intelligence*, pages 466–475.

Liang, M., Wang, W., Dong, C., and Zhao, D. (2020). A cooperative coevolutionary optimization design of urban transit network and operating frequencies. *Expert Systems with Applications*, 160.

López-Ibáñez, M., Cáceres, L. P., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2016a). The irace package: User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004*.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016b). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 363–397. Springer US, Boston, MA.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2019). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 129–168. Springer International Publishing, Cham.

Mahey, P., Mateus, G. R., Ravetti, M. G., and Souza, M. C. (2017). Optimization in networks: Modeling, algorithms and applications, accessed jan. 28, 2022. *Pesquisa Operacional [online]*, 37(3):435–436.

Maniezzo, V., Boschetti, M. A., and Stützle, T. (2021). *Matheuristics: Algorithms and Implementations*. EURO Advanced Tutorials on Operational Research. Springer, Cham, 1 edition.

Marsten, R. E. and Shepardson, F. (1981). Exact solution of crew scheduling problems using the set partitioning model: Recent successful applications. *Networks*, 11(2):165–177.

Mesquita, M., Moz, M., Paias, A., and Pato, M. (2013). A decomposition approach for the integrated vehicle-crew-roster problem with days-off pattern. *European Journal of Operational Research*, 229(2):318–331.

Mesquita, M. and Paias, A. (2008). Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers and Operations Research*, 35(5):1562–1575.

Mingozzi, A., Boschetti, M. A., Ricciardelli, S., and Bianco, L. (1999). A set partitioning approach to the crew scheduling problem. *Operations Research*, 47(6):873–888.

Osaba, E., Villar-Rodriguez, E., Del Ser, J., Nebro, A. J., Molina, D., LaTorre, A., Suganthan, P. N., Coello Coello, C. A., and Herrera, F. (2021). A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. *Swarm and Evolutionary Computation*, 64:100888.

PassMark Software Pty Ltd (1998). Cpu benchmark website, accessed jan. 18, 2021. `https://www.cpubenchmark.net`.

Patrikalakis, I. and Xerocostas, D. (1992). A new decomposition scheme of the urban public transport scheduling problem. In *Computer-Aided Transit Scheduling: Proceedings of the Fifth International Workshop*, pages 407–425.

Pepin, A., Desaulniers, G., Hertz, A., and Huisman, D. (2008). A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, 12:17–30.

Perumal, S. S. G., Dollevoet, T., Huisman, D., Lusby, R. M., Larsen, J., and Riis, M. (2021). Solution approaches for integrated vehicle and crew scheduling with electric buses. *Computers & Operations Research*, 132:105268.

Puchinger, J. and Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In Mira, J. and Álvarez, J. R., editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, pages 41–53, Berlin, Heidelberg. Springer Berlin Heidelberg.

Raidl, G. R. (2015). Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244(1):66–76.

Saha, J. L. (1970). An algorithm for bus scheduling problems. *Operational Research Quarterly*, 21:463–474.

Simões, E. M. L., Mateus, G. R., and Souza, M. J. F. (2011). Algoritmo para programação integrada de veículos e tripulações no sistema de transporte público por ônibus. In *XLIII Simpósio Brasileiro de Pesquisa Operacional*, pages 1459–1471, Ubatuba-SP.

Sörensen, K., Sevaux, M., and Glover, F. (2018). A history of metaheuristics. In Martí, R., Pardalos, P. M., and Resende, M. G. C., editors, *Handbook of Heuristics*, pages 791–808. Springer International Publishing, Cham.

Steinzen, I. (2007a). *Instances for integrated vehicle and crew scheduling problems with multiple depots, accessed Sept. 6, 2019.* `http://dsor.uni-paderborn.de/index.php?id=bustestset&L=0`.

Steinzen, I. (2007b). Topics in integrated vehicle and crew scheduling in public transport. Unpublished doctoral dissertation, University of Paderborn, Paderborn, Germany.

Steinzen, I., Becker, M., and Suhl, L. (2007). A hybrid evolutionary algorithm for the vehicle and crew scheduling problem in public transit. In *2007 IEEE Congress on Evolutionary Computation*, pages 3784–3789.

Steinzen, I., Gintner, V., Suhl, L., and Kliewer, N. (2010). A time-space network approach for the integrated vehicle- and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3):367–382.

Stützle, T. and Ruiz, R. (2018). Iterated local search. In Marti, R., Pardalos, P., and Resende, M., editors, *Handbook of heuristics*, pages 579–605. Springer, Cham.

Tahir, A., Desaulniers, G., and El Hallaoui, I. (2019). Integral column generation for the set partitioning problem. *EURO Journal on Transportation and Logistics*, 8(5):713–744.

Talbi, E.-G. (2016). Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research*, 240(1):171–215.

Wolsey, L. A. (2020). *Integer programming*. John Wiley & Sons, Ltd.

Wren, A. (1972). Bus scheduling: an interactive computer method. *Transportation Planning and Technology*, 1:115–122.

Wren, A. (1981). *Computer Scheduling of Public Transportation: Urban Passenger Vehicle and Crew Scheduling*. Elsevier Science Inc.